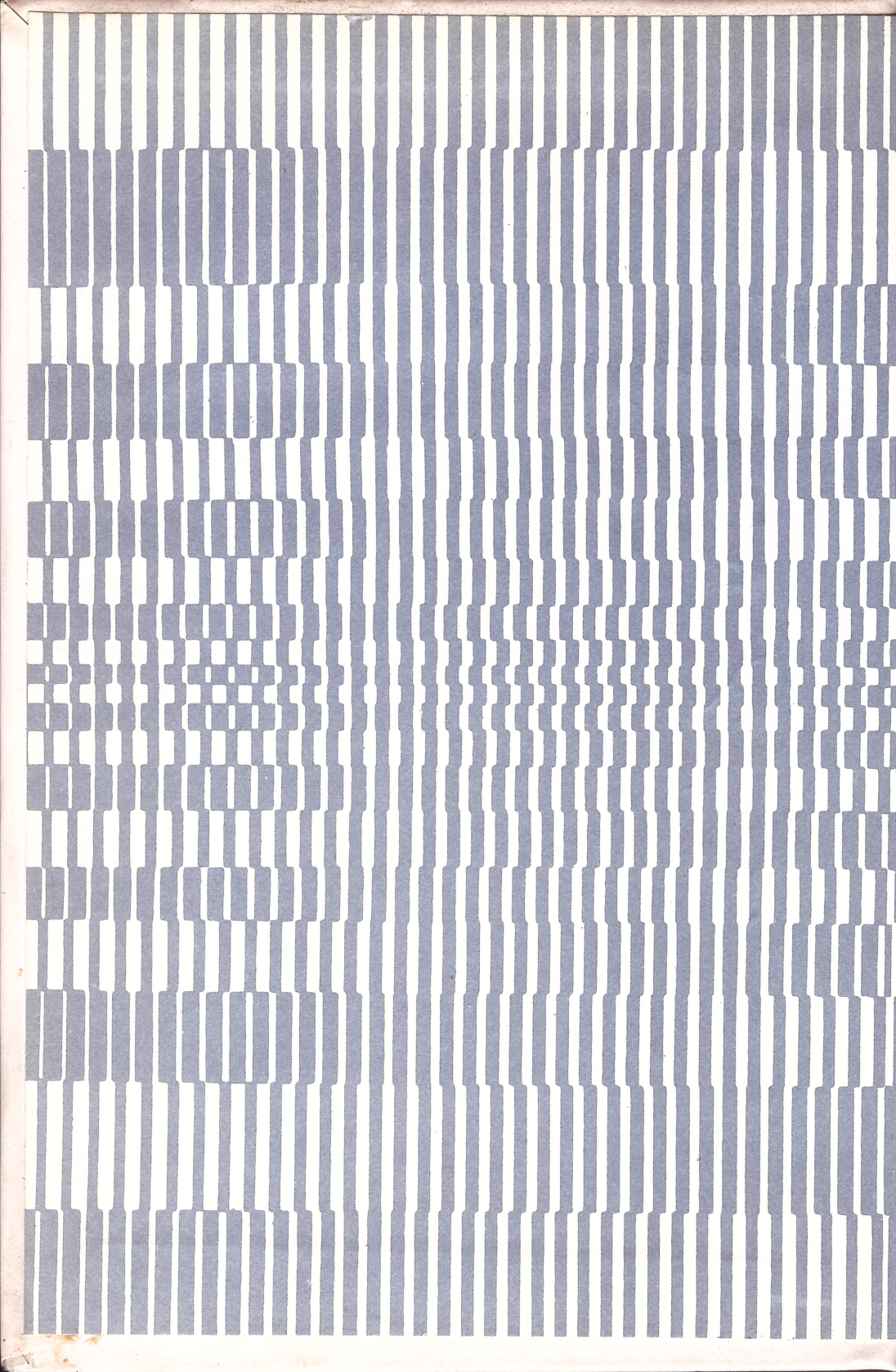
ПРОЕКТИРОВАНИЕ СИСТЕМ НА МИКРО-ПРОЦЕССОРАХ

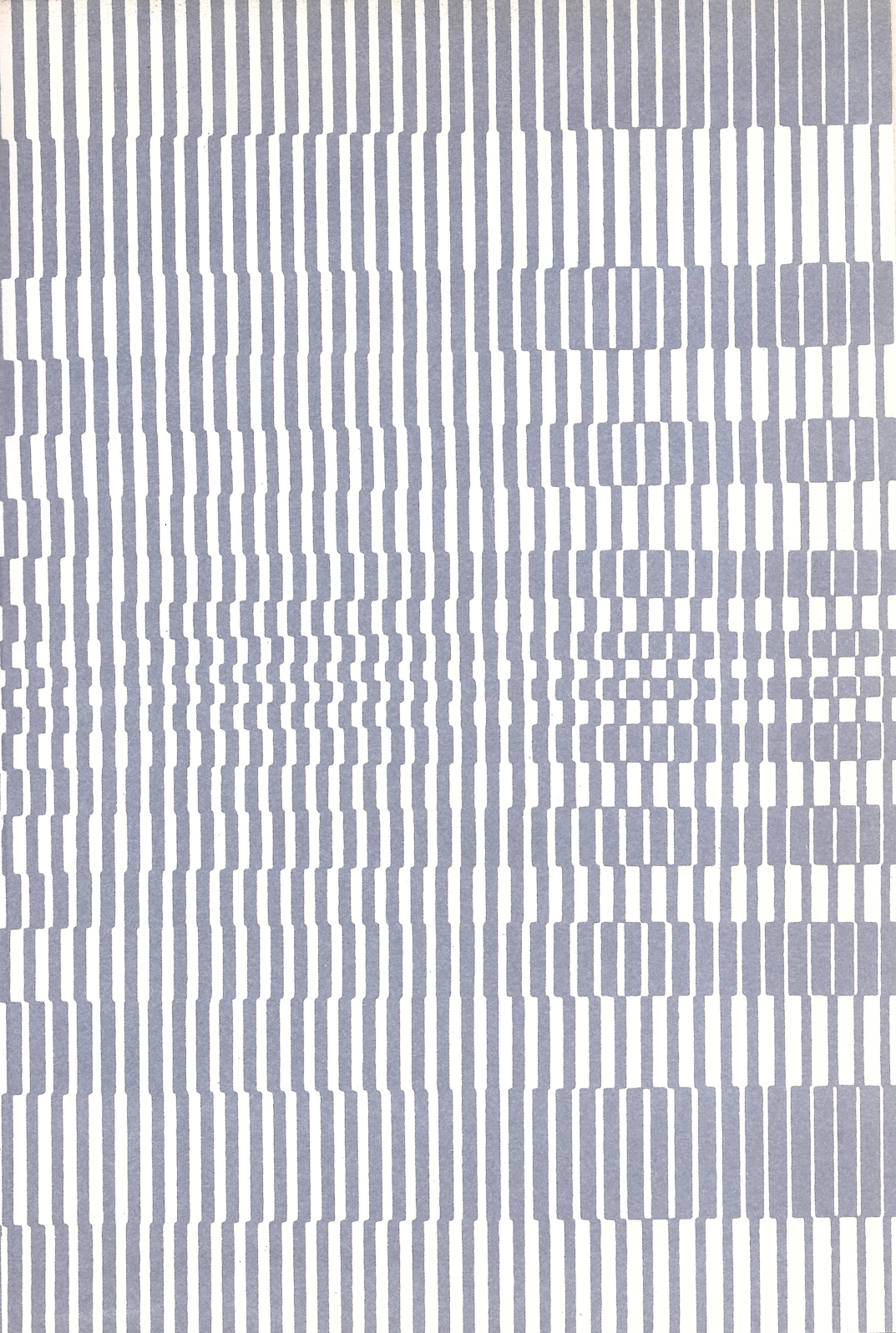
Д. ЗИССОС

ПРОЕКТИРОВАНИЕ СИСТЕМ НА МИКРО-ПРОЦЕССОРАХ

Д. ЗИССОС

Д. ЗИССОС





Д. ЗИССОС

ПРОЕКТИРОВАНИЕ СИСТЕМ НА МИКРО-ПРОЦЕССОРАХ

Перевод с английского А. И. Петренко,
В. Ю. Лапия, В. Г. Артюхова, С. М. Молявко
под редакцией проф. А. И. Петренко

Киев „Техніка“ 1982

ББК 32.973.2
6Ф7.3
З-64

Зиссос Д.
З-64 Проектирование систем на микропроцессорах / Пер. с англ. под ред. А. И. Петренко.— К.: Техніка, 1982.— 176 с., ил.
В пер.: 1 р. 10 к. 10 000 экз.

Изложены методы проектирования устройств и систем с применением микропроцессоров (МП). Материал представлен на новейшей серии микропроцессоров. Приведено большое количество схем, справочных таблиц и примеров, иллюстрирующих описанные алгоритмы проектирования систем на МП.

Рассчитана на инженерно-технических работников, занимающихся разработкой и эксплуатацией систем на микропроцессорах, а также может быть полезной студентам вузов соответствующих специальностей.

$\frac{30502\text{-}004}{З\ M202(04)\text{-}82}$ 78.82 2405000000

ББК 32.973.2
6Ф7.3

Редакция литературы по энергетике, электронике, кибернетике и связи
Зав. редакцией З. В. Божко

System design with microprocessors
D. Zissos
Department of Computer Science, University of Calgary, Calgary, Canada with contributions by J. C. Bathory, Research Assistant, University of Calgary
1978
AP
Academic Press
London New York San Francisco
A Subsidiary of Harcourt Brace Jovanovich, Publishers

© Перевод на русский язык, «Техніка», 1982
Copyright © 1978 by Academic Press Inc. (London) LTD

ОТ РЕДАКТОРА ПЕРЕВОДА

Наиболее значительным событием последних лет в области цифровой электроники является разработка микропроцессорных БИС. Объединение микросхем микропроцессора, памяти и устройств ввода/вывода позволяет построить микропроцессорную систему (например, микро-ЭВМ), которая способна заменить не только мини-ЭВМ, но в целом ряде новых областей и большую ЭВМ. Уже сейчас микропроцессорные системы применяются в периферийном оборудовании, бытовой технике, торговых терминалах, обучающих устройствах, медицинском оборудовании, автомобилях и др. По мере расширения сферы использования микропроцессоров возникает необходимость ознакомления большого контингента лиц — неспециалистов в данной области — с основными принципами организации и проектирования микропроцессорных систем. Применение микропроцессорных систем требует знания не только их технических средств, но и программного обеспечения (ПО).

Книга профессора Д. Зиссоса университета г. Калгари (Канада) «Проектирование систем на микропроцессорах», изданная в 1978 г. издательством Академии Прэсс, предназначена для всех, кто связан с разработкой, использованием и эксплуатацией систем на микропроцессорах. Она доступна лицам, не имеющим большого опыта работы в данной области.

В отличие от имеющихся отечественных и переводных книг по микропроцессорам и микро-ЭВМ (например, Д. Хилбурна, П. Джулича «Микро-ЭВМ и микропроцессоры», «Мир», 1979; Б. Соучека «Микропроцессоры и микро-ЭВМ», «Сов. радио», 1980 и др.), содержащих, как правило, базовые сведения о структурах, командах и параметрах отдельных конкретных микропроцессоров и микро-ЭВМ, книга Д. Зиссоса посвящена вопросам проектирования интерфейсов различных типов микропроцессорных систем, например, систем типа ожидания/счета,

синхронизирующих машинный цикл микропроцессора с реакцией внешнего устройства. В книге Д. Зиссоса рассмотрены процедуры проектирования на конкретных примерах логических схем для микропроцессоров *Intel 8080* и *Motorola 6800* (например, в задачах обеспечения стартстопного режима фотосчитывателя и печатающего устройства).

Далее описаны системы типа проверки и пропуска, также синхронизирующие циклы микропроцессора и внешнего устройства программными средствами. На примерах организации передачи данных (например, от ОЗУ к печатающему устройству, от фотосчитывателя через память на печать и др.) показаны процедуры проектирования технических средств и программного обеспечения систем рассматриваемого типа.

Следующие главы посвящены изложению методики проектирования систем прерывания в микропроцессорных системах на базе *Intel 8080 и Motorola 6800* (на примерах формирования восьмибитовой печатной строки и связи ОЗУ с печатающим устройством), а также систем прямого доступа в память (ПДП) с соответствующими интерфейсами и логическими схемами «захвата» цикла (на примерах организации передачи данных между памятью микропроцессора и внешними устройствами).

В книге рассмотрены системы прямой передачи данных (ППД), позволяющие осуществить передачу данных непосредственно между внешними устройствами (на примерах организации копирования перфоленты, копирования ленты с устранением ошибок), и системы типа состояние и действие, которые позволяют управлять внешними устройствами с гораздо меньшим числом сигналов управления.

Настоящая книга, в которой последовательно, методически четко и доступно излагаются методы проектирования интерфейсов наиболее распространенных типов для микропроцессорных систем, принесет несомненную пользу советским читателям. Однако книга Д. Зиссоса не исчерпывает многообразие острых и важных вопросов, стоящих перед разработчиками современных микропроцессорных систем. Ее материал хорошо сочетается с ранее изданными книгами по программированию микропроцессоров и общим вопросам микропроцессорных систем.

Перевод книги выполнен проф. А. И. Петренко, докт. техн. наук В. Ю. Лапием, канд. техн. наук В. Г. Артюховым и С. М. Молявко.

Проф. А. И. Петренко

ПРЕДИСЛОВИЕ АВТОРА

Возможности микропроцессоров в настоящее время широко известны. Однако не всегда подчеркивается, что эти приборы, объединяющие в себе простоту и универсальность, могут использоваться лицами, не знакомыми с электроникой, для построения своих систем. Такие пользователи, однако, должны располагать формальными процедурами поэтапного проектирования, и данная книга как раз и предназначена для тех, кто желает создавать микропроцессорные системы. Цель книги — уберечь микропроцессоры от судьбы больших ЭВМ 50—60-х годов, когда основное внимание уделялось разработке программ для ЭВМ и формализованные процедуры аппаратурных реализаций оставались в стороне. В результате возникла отрасль, влияющая на судьбу миллионов людей, но доступная лишь немногим. Компьютеры представляют собой тайну, покрытую мраком, и этот мрак сгущается.

Первая часть книги вводит читателя в базовую методику проектирования логических схем, необходимую для разработки и построения микропроцессорных систем. Во второй части описывается микропроцессорный (МП) кристалл, выделяются общие для различных МП систем характеристики, а также формулируется методика проектирования, развиваемая в этой книге. В дальнейшем описываются различные МП системы, с помощью которых решаются различные задачи. Среднее время разработки малой МП системы в настоящее время составляет около 50 мин. Нет необходимости выделять время на отладку системы, так как система, спроектированная по изложенной методике, всегда работает.

Желающие получить более подробную информацию о программировании МП систем могут обратиться к последней книге Ф. Дункана «Программирование МП и конструирование программ», изданной издательством Прентис-Холл в 1979 г. Исследования, на результатах которых базируется данная книга, проводились при финансировании Национального исследовательского совета Канады.

Калгари, август, 1978, Д. Зиссос

ВВЕДЕНИЕ

Влияние новой технологии на ЭВМ

Не может быть двух мнений о том, что современное развитие микроэлектроники, обсуждаемое в данной книге, оказывает сильное влияние на ЭВМ, но до сих пор нет мнения о характере этого влияния. Группа по ЭВМ университетских и исследовательских центров, санкционирующая и поддерживающая все крупнейшие исследования по вычислительной технике в университетах, весьма заинтересована во внедрении новой элементной базы и технологии ЭВМ. Создана рабочая группа под моим руководством для оценки сложности работ в этой области и определения их влияния на вычислительную технику, а также для оказания помощи и обеспечения разработок. Рабочая группа приступила к своей деятельности, но я хочу прокомментировать впечатления и мнения, которые возникли у меня в ходе интенсивных дискуссий, предшествующих началу работы группы.

Очень важно понимать перспективу развития данной тематики. Часто публикуемые в области вычислительной техники работы содержат утверждения об отмирании вычислительных центров и приближающемся господстве сетей ЭВМ. Сторонники этой точки зрения предсказывают быстрое проникновение распределенных структур ЭВМ во все организации, отделы и даже к отдельным пользователям, которые будут располагать мощными вычислительными системами на своих рабочих местах. Я бы хотел отметить, что пока будут проявляться сдвиги в области создания распределенных сетей ЭВМ, значительно более важные изменения будут наблюдаться в проникновении ЭВМ в новые для них области, во внедрении ЭВМ во все сферы деятельности общества.

Первым и наиболее важным моментом является факт одинаковой важности аппаратурного и программного обеспечения ЭВМ. Данная книга одинаково хорошо освещает оба аспекта. Впечатляющее снижение стоимости вычислительной мощности, основной памяти и внешних ЗУ сопровождается резким возрастанием

стоимости программ и обслуживания. Уже сегодня возрастающая стоимость программного обеспечения и обслуживания намного превышает стоимость аппаратуры. Сейчас стоимости программного обеспечения и аппаратурных средств для больших и средних ЭВМ, усредненные на периодах их эксплуатации, составляют 80 и 20% соответственно. Даже в предельном случае при бесплатной передаче аппаратуры общая стоимость вычислительных систем снизится примерно на 20%. Таким образом, стоимость больших и средних ЭВМ, эксплуатируемых в традиционных вычислительных центрах, мало зависит от внедрения новой технологии. Как же новая технология влияет на стоимость распределенных сетей ЭВМ?

Важно отметить, что современная микроэлектроника проникает в общественное производство двумя путями: путем больших количеств микропроцессоров и микро-ЭВМ малой стоимости и малых количеств крупных ЭВМ большой стоимости.

Первый путь — это производство больших количеств дешевых приборов. Чаще всего — это просто одиночные кристаллы, ориентированные на конкретные применения, и поэтому они не могут рассматриваться как компьютеры. Такие кристаллы (контроллеры) можно применять в обычной аппаратуре и в таком оборудовании, как автомобили, стиральные машины, телевизоры. Они не имеют периферии (кроме сигнальных преобразователей) и не требуют обслуживания, необходимого для обычных ЭВМ. Большие количества поставок требуются для снижения себестоимости таких контроллеров. Широкое внедрение в новые области жизненно важно для новой МП промышленности, так как существующий рынок средств обработки данных не в состоянии обеспечить ее рост. По существующим оценкам, мировые потребности индустрии средств обработки в 1978 г. могут быть удовлетворены производителями микропроцессоров за две недели. Безусловно, рынок средств обработки данных будет успешно развиваться в связи с появлением новых приложений микропроцессоров, но это развитие не будет решающей силой. Разработки нового оборудования в большей степени будут опираться на поставляемые коммерческие кристаллы, чем на собственные «заказные» микросхемы.

Аппаратура обработки данных станет основным потребителем крупных ЭВМ высокой стоимости, выпускаемых в малых количествах. Системы высшего

уровня будут строиться на кристаллах при соответствующем увеличении их цен. Такие системы с точки зрения электронной части аппаратуры дешевы, но дорогие механические периферийные устройства будут увеличивать их цену. Новая технология, однако, обеспечивает создание таких специализированных устройств, как матричные процессоры и системы управления базами данных, превосходящие по характеристикам современные системы. В будущем пользователям станут доступны различные системы от настольных ЭВМ стоимостью несколько сот фунтов до больших систем с производительностью 50 млн. операций в секунду и стоимостью примерно 2 млн. фунтов. Таким образом, по аппаратурным затратам новая технология существенно снизит цены.

Но ЭВМ состоит не только из аппаратуры. Необходимое для работы программное обеспечение (ПО) также представлено двумя категориями: дешевого ПО, тиражируемого в больших количествах, и уникального ПО высокой стоимости, каждое из которых работает на аппаратуре соответствующей категории (контроллерах и крупных ЭВМ). ПО для контроллеров специального назначения также является узкоориентированным, относительно свободным от ошибок и весьма недорогим из-за больших объемов поставок, оно не изменяется пользователем и не требует сопровождения. ПО высокой стоимости, главным образом, предназначено для средств обработки данных. Относительно малые объемы поставок неизбежно ведут к высоким ценам. Из-за высоких цен пользователи требуют сопровождения программ, что ведет к дальнейшему повышению их цен. Таким образом, конкретный пользователь сталкивается не только со снижением стоимости аппаратуры, но и с гораздо более стабильным повышением стоимости ПО. До тех пор, пока существуют эти две тенденции, эра дешевых ЭВМ, предсказываемая многими, просто не наступит.

Однако имеются некоторые потенциальные опасности в применении новой элементной базы при разработке ЭВМ. Некоторые пользователи предпочитают покупать недорогую аппаратуру и разрабатывать свое собственное ПО. В настоящее время это рискованный подход, так как наблюдаются большие изменения в области ПО. Доступное для современных микропроцессоров ПО примитивно, как правило, написано на ассемблерном или машинном уровне. При попытке автономной разработки ПО потребуется большое

количество дополнительного оборудования и программ для отладки: эмуляторы, отладчики, логические анализаторы и системы отладки. В этом случае превышение стоимости конечной системы в 5—10 раз по сравнению с исходной за счет сопутствующего оборудования не будет исключением. Кроме того, еще и ПО потребует затрат, так как его разработка проводится с участием человека. Однако существующие сейчас 16-битовые микропроцессорные системы будут через несколько лет вытеснены 32-битовыми, в результате обесценятся все купленные ранее отладочные средства. Имеются, безусловно, области, требующие самостоятельных значительных разработок ПО, но не следует недооценивать возникающие при этом трудности, и я надеюсь, что читатели этой книги — разработчики ПО — уделяют в своей работе достаточно внимания сделанным предостережениям.

Существуют и другие причины высокой стоимости ЭВМ, даже в распределенных сетях. Одна из них — организация файлов, являющихся основой любой системы обработки данных. В настоящее время быстрое развитие дисковых систем файлов делает распределенное хранение файлов неэкономичным. Диски емкостью в биллон байт, появившиеся около двух лет назад, не могут быть заменены наборами гибких дисков или магнитных запоминающих устройств на цилиндрических доменах. Но организация файлов не определяется только стоимостью внешнего ЗУ. Вычислительный центр с большой центральной памятью для файлов позволяет организовать хранение множества файлов значительно дешевле, чем распределенные терминалы пользователей сети ЭВМ, кроме того, надежность и секретность хранения файлов при централизованной организации (один или несколько центров) значительно выше. Единственный случай, когда целесообразна распределенная система хранения файлов, соответствует случаю полной независимости файлов отдельных пользователей. Если же их зависимость даже предельно минимальна, значительно возрастает стоимость передачи данных в распределенной сети ЭВМ.

Я придерживаюсь мнения, что новые компьютеры будут организованы по иерархической схеме со связью через сеть. Степень сосредоточенности вычислительной мощности и памяти для файлов в отдельных узлах будет зависеть больше от стоимости передачи данных, чем от стоимости микропроцессорного оборудования. Любое изменение в стоимости передачи данных

приведет к изменению в распределенности сети. С этой точки зрения в будущем будут использоваться вычислительные центры, сети ЭВМ и отдельные ЭВМ. Основной результат появления новой технологии, однако, будет состоять в проникновении ЭВМ в новые сферы и отрасли, а также в увеличении количества людей, использующих ЭВМ. Расширение круга пользователей за счет людей, не сведущих в ЭВМ, ставит новые требования для техники программирования. Новое ПО должно быть значительно лучше приспособлено к человеческим способностям и больше ориентировано на пользователей. Я называю такое ПО «симпатичным». По счастливому стечению обстоятельств, та самая технология, которая создает потребность в таком ПО, обеспечивает экономичную базу для его разработки. В прошлом очень часто программист был ограничен в использовании основной памяти ЭВМ. Теперь он не только получает дополнительную память малой стоимости, но и может создавать различные по степени детализации интерпретаторы для разных классов пользователей. Кроме того, благодаря снижению стоимости интерактивных систем, появляется возможность интерактивного диалога с пользователем — непрофессионалом для устранения неточности интерпретации ответов. Кроме того, такое ПО может содержать «службу помощи» для сопровождения пользователя в запутанных ситуациях.

Я надеюсь, что в этом предисловии я обратил Ваше внимание на те широкие перспективы, которые открываются введением новой микротехники и технологии, описанной в прекрасной книге проф. Д. Зиссоса. Давайте же сосредоточим наше внимание на большей наглядности представления ЭВМ новым пользователем, подробно рассматривая основные этапы разработки «симпатичного» ПО. Если мы не реализуем эту задачу, то не следует ожидать быстрого вторжения ЭВМ в новые области применения.

Ливерпуль, август, 1978, И. Алти

ГЛАВА 1

ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

В этой главе рассмотрены основные положения, необходимые для понимания и приобретения навыков проектирования и конструирования аппаратурной части систем.

Описанные шаги проектирования базируются на использовании последовательностных уравнений, разработанных автором в 1969 г. Все схемы, реализующие эти уравнения, свободны от гонок (состязаний сигналов) при условии использования вентилей с разбросом задержек сигнала до $33^1/_3\%$ от максимального значения.

1.1. БАЗОВЫЕ ОПРЕДЕЛЕНИЯ

Логическое проектирование определяется как набор детерминированных последовательных процедур, используемых на практике при реализации логических схем по их входным и выходным характеристикам. Логические схемы подразделяются на комбинационные и последовательностные. Комбинационная — это схема, для которой выходные сигналы являются функциями входных, в то время как для последовательностных схем выходные сигналы определяются последовательностью поступления входных. Последовательностные схемы обладают памятью. Примером комбинационной схемы может служить домашнее освещение, управляемое выключателем. Когда выключатель замкнут, свет горит, когда разомкнут,— погашен. Осветительная сеть, управляемая кнопочным выключателем, является последовательностной, так как результат нажатия кнопки зависит от предыдущего состояния цепи. Если свет горит, нажатие кнопки выключает его, и наоборот.

В свою очередь, последовательностные схемы подразделяются на асинхронные (нетактируемые) и синхронные (тактируемые) *.

Нетактируемые, или асинхронные, схемы непосредственно реагируют на входные сигналы в отличие от синхронных схем, реакция которых определяется, кроме того, воздействием тактового сигнала, причем при отсутствии тактового сигнала

* Третий вид последовательностных схем, названный «цепями, управляемыми импульсами», рассмотрен во 2-м издании книги Д. Зиссоса «Задачи логического проектирования и их решения». Оксфорд, Юниверсити Пресс, 1978.

состояние такой схемы остается неизменным. При аппаратурной реализации асинхронной схемы можно использовать только логические элементы, в то время как для реализации синхронной схемы требуются запоминающие элементы.

До 1969 г., пока не были разработаны последовательностные булевы уравнения, логическое проектирование асинхронных схем основывалось на эмпирических принципах, уделявших мало внимания инженерной реализации системы вплоть до конечного этапа. Появление последовательностных уравнений дало возможность создания детерминированных последовательных алгоритмов, которые учитывают реальное воплощение схемы на всех стадиях проектирования. Никакие другие инженерные знания при таком проектировании не требуются.

1.2. ОПТИМАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Главная цель такого проектирования — обеспечение читателя простыми и надежными методами проектирования, понятными не только разработчику, но и пользователю. При этом элегантное решение задачи можно получить без чрезмерных усилий.

1.3. БУЛЕВА АЛГЕБРА

Необходимым условием для логического проектирования является знание булевой алгебры.

Булева алгебра — общепринятая система обозначений, в которой переменные и операторы комбинируются в выражения. Булевы переменные могут принимать одно из двух значений: «0» или «1». Это не арифметический ноль и единица. Так, например, логический «0» не означает количественно «ничто». Его можно использовать для индикации одного из двух состояний устройства с двумя состояниями (бистабильного элемента), такого как триггер или реле; второе состояние при этом обозначается булевой переменной «1». Существует большое количество логических операторов: *И, ИЛИ, НЕ, НЕТ, ИЛИ-НЕ, И-НЕ, исключающее ИЛИ* и т. д., однако для проектирования достаточно только трех — все остальные можно выразить через них. Этими операторами являются: булево сложение (дизъюнкция) *ИЛИ*, булево умножение (конъюнкция) *И*, булева инверсия (отрицание) *НЕ*.

Сложение (дизъюнкция) обозначается знаком «+». Иногда используются знаки $\vee$, $\cup$, *ИЛИ*. Выражение $A+B$ читается как *A* ИЛИ *B*, *A* плюс *B*, при этом $A+B$ истинно, если или *A*, или *B*, или оба вместе «истинны», в противном случае $A+B$ «ложно». Таким образом:

$$0+0=0; \qquad 1+0=1;$$
$$0+1=1; \qquad 1+1=1;$$

Логическое умножение обычно обозначается знаками «·» или «×». Часто этот знак опускается (как и в обычной алгебре), но при этом каждая переменная должна обозначаться одной буквой. Иногда этот знак записывается как $\wedge$, $\cap$, *И*. Выражение $A \cdot B$ читается как *A И B*, или *A* умножить на *B*. При этом $A \cdot B$ «истинно», лишь когда *A* «истинно» и *B* «истинно», в противном случае $A \cdot B$ «ложно». Другими словами,

$$0 \cdot 0 = 0; \qquad 1 \cdot 0 = 0;$$
$$0 \cdot 1 = 0; \qquad 1 \cdot 1 = 1.$$

Логическое отрицание (дополнение до 1) обозначается черточкой над его аргументом или знаком «$\neg$» перед ним. Иногда оно записывается как *НЕ*. Итак, инверсия *A* это $\bar{A}$ или $\neg A$, или *НЕ A*.

Булевы теоремы [1, 2]. Для наших целей, т. е. для проектирования и реализации цифровых схем, рассмотрим три теоремы. Теорему, позволяющую исключить избыточность схемы; теорему, подавляющую нежелательные гонки, и теорему де Моргана. Первую назовем теоремой избыточности, вторую — теоремой гонок.

Теорема 1. Теорема избыточности $A + \bar{A}B = A$.

Доказательство:

$$A + AB = A(1 + B) = A \cdot 1 = A.$$

Эта теорема обозначает, что в булевых выражениях, содержащих сумму произведений, то слагаемое, которое содержит все сомножители другого слагаемого, является избыточным. Например, в булевой функции $f = AB + ABC + \bar{A}BD$ произведения ABC и $\bar{A}BD$ избыточны и их можно отбросить, так как каждое из них содержит все сомножители первого слагаемого AB.

Теорема 2. Теорема гонок $AB + \bar{A}C = AB + \bar{A}C + BC$.

Доказательство:

$$AB + \bar{A}C + BC = AB + \bar{A}C + (A + \bar{A})BC =$$
$$= AB + \bar{A}C + ABC + \bar{A}BC =$$
$$= AB(1 + C) + \bar{A}C(1 + B) =$$
$$= AB + \bar{A}C.$$

Эта теорема позволяет вводить дополнительные члены в выражение типа суммы произведений *. Дополнительные члены представляют собой произведение переменных при *A* и $\bar{A}$ в выражении $AB + \bar{A}C$.

Слагаемое BC является дополнительным до тех пор, пока породившие его члены (AB и $\bar{A}C$) остаются в выражении. Если один из породивших его членов удалить (при использовании теоремы 1), оно уже не будет дополнительным и не может быть удалено из выражения. Вышесказанное иллюстрируется тремя примерами.

* Булево произведение считается дополнительным, если его присутствие в выражении не влияет на величину функции и его можно заключить в скобки.

Пример 1. В булевом выражении $f=A+\bar{A}B$ первое слагаемое, которое можно записать как $A\cdot 1$, содержит A, а второе произведение $\bar{A}B$ содержит $\bar{A}$. Поэтому в соответствии с теоремой 2 можно ввести дополнительное произведение $1\cdot B$ и тогда

$$f=A+\bar{A}B+(B).$$

Из теоремы 1 следует, что произведение $\bar{A}B$ избыточно, так как оно содержит все элементы (в данном случае просто B) произведения B. Поскольку член $\bar{A}B$ (один из породивших B) теперь не входит в выражение, член B не является дополнительным.

Схематически эти шаги можно представить в виде

$$f=A+\bar{A}B$$

B замещает $\bar{A}B$,

$$=A+B$$ — окончательный результат.

Пример 2. Рассмотрим булево выражение $f=AB+\bar{A}C+BCD$. Наличие A в произведении AB и $\bar{A}$ в произведении $\bar{A}C$ с учетом теоремы 2 позволяет ввести дополнительный член BC. Таким образом, $f=AB+\bar{A}C+BCD+(BC)$.

Согласно теореме 1, произведение BCD является избыточным, так как оно содержит все члены произведения BC, т. е. $f=AB+\bar{A}C+(BC)$.

Далее, так как члены, породившие BC, а именно AB и $\bar{A}C$, все еще присутствуют в выражении, член BC является избыточным и может быть удален; в результате получим $f=AB+\bar{A}C$.

Схематически эти шаги можно представить в виде

$$f=AB+\bar{A}C+BCD$$

BC устраняет член BCD,

$$=AB+\bar{A}C$$ — окончательный результат.

Пример 3. Рассмотрим булево выражение $f=A+\bar{A}B+BC$. Дополнительный член B, порождаемый первыми двумя членами A и $\bar{A}B$, замещает член $\bar{A}B$ и подавляет оставшийся член BC. Схематически это можно представить в виде

$$f=A+\bar{A}B+BC$$

B замещает породивший его $\bar{A}B$ и подавляет BC,

$$=A+B$$ — окончательный результат.

Таким образом, дополнительный член можно использовать для удаления не участвовавшего в порождении члена булевого выражения и (или) для замещения членов, участвовавших в его порождении.

Теорема 3. Теорема де Моргана. Дополнение булева выражения можно получить прямой заменой каждой переменной ее дополнением в соответствующем дуальном выражении. Например, дуальным к выражению $P=A+BC$ является выражение $A\cdot(B+C)$. Таким образом, по теореме де Моргана дополнением P является $\bar{P}=\bar{A}\cdot(\bar{B}+\bar{C})$.

Доказательство: пусть выражение P подлежит инвертированию, а выражение, полученное заменой каждой переменной

в дуальном к P выражении, на ее дополнение обозначим Q. Необходимо доказать, что $Q=\overline{P}$. Это может быть тогда и только тогда, когда $P\cdot Q=P\cdot\overline{P}=0$ и $P+Q=P+\overline{P}=1$.

Рассмотрим следующие случаи.

1. Пусть P — простая константа (0 или 1) или переменная, например, $P=A$, тогда $Q=\overline{A}=\overline{P}$. Далее, если $P=\overline{A}$ (инвертированная переменная), то $Q=\overline{A}=A=\overline{P}$.

2. Пусть P — сумма двух слагаемых A и B, которые сами могут быть выражениями. Тогда $P=A+B$; $Q=\overline{A}\,\overline{B}$.

Далее

$$PQ=(A+B)\,\overline{A}\,\overline{B}=A\overline{A}\,\overline{B}+\overline{A}\,\overline{B}B=0+0=0$$

и

$$\begin{aligned}P+Q&=A+B+\overline{A}\,\overline{B}=A+B+\overline{A}\overline{B}+\overline{B}= && \text{(теорема 2)}\\ &=A+B+\overline{B}= && \text{(теорема 1)}\\ &=A+1=\\ &=1.\end{aligned}$$

Значит, $Q=\overline{P}$.

3. Предположим, P является произведением двух членов A и B, которые сами могут быть выражениями. Тогда $P=AB$; $Q=\overline{A}+\overline{B}$

и

$$\begin{aligned}PQ&=AB\,(\overline{A}+\overline{B})=\\ &=0+0=0;\end{aligned}$$

$$P+Q=AB+\overline{A}+\overline{B}=AB+\overline{A}+\overline{B}+B=1.$$

Следовательно, $Q=\overline{P}$.

Прежде чем инвертировать заданное выражение, целесообразно упростить выражение и заключить все члены произведения в скобки. Скобки инвариантны к процессу дополнения.

Пример 4. Найти дополнение к $P=A+B\overline{C}$. Предлагаемая процедура.

Исходное выражение $P=A+B\overline{C}$

Минимизированное $P=A+B\overline{C}$

Скобочная форма $P=A+(B\overline{C})$

Инверсия $\overline{P}=\overline{A}(\overline{B}+C)$

Удаление избыточных скобок $\overline{P}=\overline{A}(\overline{B}+C)$.

Пример 5. Найти дополнение к $f=A(BC+\overline{B}\overline{C}+BCD)$. Предлагаемая процедура.

Исходное выражение $f=A(BC+\overline{B}\overline{C}+BCD)$

Минимизированное $f=A(BC+\overline{B}\overline{C})$

Скобочная форма $f=A\,[(BC)+(\overline{B}\overline{C})]$

Инверсия $\overline{f}=\overline{A}+[(\overline{B}+\overline{C})(B+C)]$

Удаление избыточных скобок $f=\overline{A}+(\overline{B}+\overline{C})(B+C)$.

Пример 6 (задача о болтушках).

Дано:

(1) Алиса никогда не болтает;

(2) Бетти болтает только в присутствии Алисы;

(3) Клара болтает в любых условиях, даже когда она одна;
(4) Доротти болтает тогда и только тогда, когда рядом Алиса.
Определить, когда в комнате царит молчание.

Решение. Пусть $G=1$ соответствует условию, когда в комнате болтают; таким образом, условие $G=0$ справедливо для комнаты, в которой тишина.

Пусть $A=1$ обозначает присутствие, а $A=0$ — отсутствие Алисы. Аналогично B, K, D относятся к Бетти, Кларе и Дороти соответственно. Переводя указанные условия в форму булевого уравнения, имеем

$$G = AB + K + AD$$

(слагаемые расположены в порядке их получения из условий 2, 3, 4; условие 1 порождает слагаемое $A\cdot 0=0$).

Для определения $\overline{G}$ (условие тишины в комнате) выполним следующие преобразования:

Исходное выражение $G = AB + K + AD$
Минимизированное $G = AB + K + AD$
Скобочная форма $G = (AB) + K + (AD)$
Инверсия $\overline{G} = (\overline{A} + \overline{B})\,\overline{K}\,(\overline{A} + \overline{D})$
Удаление избыточных скобок $\overline{G} = \overline{A}\,\overline{K} + \overline{B}\,\overline{K}\,\overline{D}$.

Таким образом, в комнате будет тишина, если отсутствуют Алиса и Клара или отсутствуют Бетти, Клара и Доротти.

Прежде чем инвертировать заданное выражение, целесообразно его редуцировать и заключить произведение в скобки. Скобки инвариантны по отношению к операции дополнения.

Пример 7. Найти дополнение к $P=A+B\overline{C}+AD$. Предлагаемая процедура.
Исходное выражение $P=A+B\overline{C}+AD$
Редуцированное (упрощенное) $P=A+B\overline{C}$
Скобочная форма $P=(A)+(B\overline{C})$
Инверсия $\overline{P}=(\overline{A})(\overline{B}+C)$
Удаление избыточных скобок $\overline{P}=\overline{A}(\overline{B}+C)$.

Булева редукция. Булева функция называется редуцированной, если не содержит дополнительных членов, присутствие которых не сказывается на значении функций. Например, сомножитель $\overline{A}$ в $A+\overline{A}B$ является избыточным, так как $A+\overline{A}B=A+B$. Редуцирование в двухуровневых булевых выражениях может проводиться в три этапа с использованием теорем 1 и 2. Если выражение содержит более двух уровней, как в $f=A+B(C+D)$, его необходимо перевести в двухуровневую форму (в сумму произведений) путем раскрытия скобок.

Ниже описаны три шага алгоритма редуцирования булевых выражений.

Шаг 1 — раскрытие скобок. Редуцируемое выражение преобразуем в двухуровневую форму (сумму произведений) путем раскрытия скобок. Произведения, которые после этого содержат как переменные, так и их дополнения, подлежат отбрасыванию по правилу $A\cdot\overline{A}=0$. Повторение переменной в произведении исключается по правилу $A\cdot A=A$. Полученные произведения располагаются в порядке возрастания числа сомножителей слева направо.

Пример 8. Рассмотрим булеву функцию $f = BC + (AB + D)\,\overline{D} + A$. Применив шаг 1, получим

$$f = BC + (AB + D)\,\overline{D} + A = BC + AB\overline{D} + D\overline{D} + A = BC + AB\overline{D} + A = A + AB\overline{D} + BC.$$

Шаг 2 — применение теоремы 1. Исключим избыточные члены, используя теорему 1. Начав с произведения с наименьшим числом сомножителей, т. е. с первого по порядку, рассмотрим каждый последующий член, имеющий большее число сомножителей (он расположен правее). Произведения, содержащие все сомножители данного члена, отбрасываются.

Пример 9. На 1-м шаге получено выражение $f = A + BC + A\bar{B}\bar{D}$. Рассматривая слагаемые слева направо, выбираем A. Далее рассматриваем члены справа от A в поисках слагаемого, содержащего A как сомножитель. Таким слагаемым является $A\bar{B}\bar{D}$, оно отбрасывается, в результате чего имеем $f = A + BC$. Так как правее BC нет других членов, 2-й шаг можно не повторять.

Шаг 3 — применение теоремы 2. На этом шаге по теореме 2 оптимизируются члены, содержащие произведения. Практически с приобретением опыта нижеописанный алгоритм сокращается. Однако в книге дано его полное описание для начинающих или желающих составить для него машинную программу.

Будем считать, что произведения расположены по возрастанию длины слева направо, как было указано ранее. Преобразования выполняются в два этапа.

1. В первом произведении выбирается первая переменная, а оставшаяся часть выражения просматривается в поисках произведения, содержащего дополнения выбранной переменной. Когда найдено такое произведение, формируется дополнительное произведение по теореме 2. Дополнительное произведение используется для исключения членов, не участвующих в его порождении, как указывалось в примере 3 после теоремы 2, или (и) для замены порождавших его членов (там же). Если порождавшие его члены заменены, дополнительный член помещается в начало выражения и 3-й шаг повторяется. Если дополнительные члены не использовались, переходим дальше.

Шаг 3 повторяется до тех пор, пока не будут выявлены все дополнительные члены первого уровня.

2. 3-й шаг повторяется, используя дополнительные члены высокого уровня. Рассмотрим применение всех трех шагов алгоритма.

Пример 10. Редуцировать $f = A + \bar{A}\bar{B} + \bar{B}DC + \bar{A}BD$.

Решение.

Шаг 1 — раскрытие скобок. Не выполняется.

Шаг 2 — применение теоремы 1. Не выполняется.

Шаг 3 — применение теоремы 2:

$f = A + \bar{A}\bar{B} + \bar{B}CD + \bar{A}BD;$

$\bar{B}$ замещает порождающее произведение $\bar{A}\bar{B}$ и исключает непорождающий член $\bar{B}CD$,

$= A + \bar{B} + \bar{A}BD;$

BD замещает порождающее произведение $\bar{A}BD$,

$= A + \bar{B} + BD;$

D замещает порождающее произведение BD,

$= A + \bar{B} + D$ — требуемый результат.

Пример 11. Привести $f = (A + C)(\bar{A} + B) + DE(\bar{B} + \bar{C}) + ABC$.

Решение.

Шаг 1 — перемножение — раскрытие скобок:

$$f = A\bar{A} + AB + \bar{A}C + BC + \bar{B}DE + \bar{C}DE + ABC = \\ = AB + \bar{A}C + BC + \bar{B}DE + \bar{C}DE + ABC.$$

Шаг 2 — применение теоремы 1

$$f = AB + \bar{A}C + BC + \bar{B}DE + \bar{C}DE.$$

ABC исключено, поскольку содержит все члены элемента AB.

Шаг 3 — применение теоремы 2:

$f = AB + \bar{A}C + BC + \bar{B}DE + \bar{C}DE;$

BC замещает непорождающее произведение BC,

$= AB + \bar{A}C + \bar{B}DE + \bar{C}DE;$

CDE

DE замещает порождающие произведения $\bar{B}DE$ и $\bar{C}DE$,

$= AB + \bar{A}C + DE$ — окончательный результат.

1.4. ЛОГИЧЕСКИЕ ЭЛЕМЕНТЫ

Элементы *И—НЕ*. Хотя логические схемы можно проектировать, используя набор элементов *И, ИЛИ, НЕ, И—НЕ, ИЛИ—НЕ, исключающее ИЛИ*, для упрощения будем применять только элементы *И—НЕ*, триггеры и трехстабильные элементы. Однако

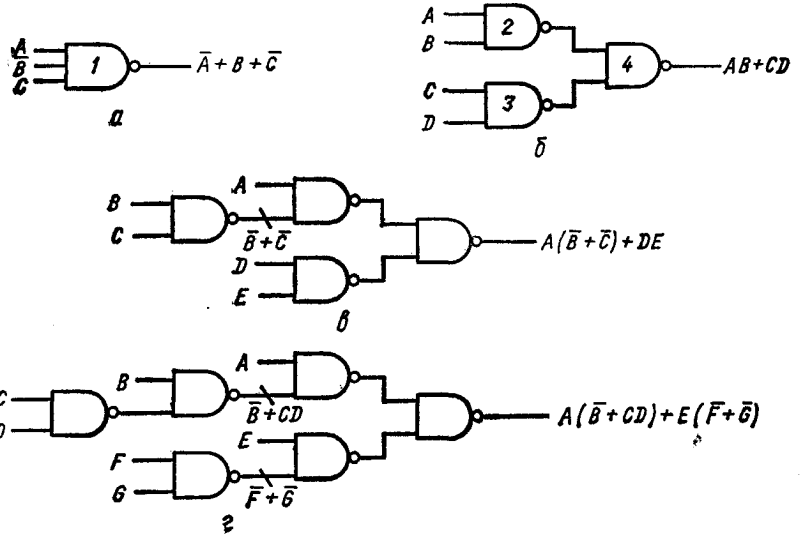


Рис. 1.1

необходимо отметить, что описываемая методика не исключает использования других элементов.

Элемент *И—НЕ* для инверсных сигналов выполняет функцию *ИЛИ*. Например, если сигналы A, $\bar{B}$, C поданы на вход элемента *И—НЕ*, на его выходе будет сигнал $\overline{A\bar{B}C} = \bar{A}+B+\bar{C}$ (рис. 1. 1, а).

Выход элемента *И—НЕ* можно описать с помощью булевого выражения типа суммы произведений по одному произведению

на каждую схему, подключенную ко входу выходного элемента. Сомножители каждого из слагаемых являются входными для соответствующих элементов. Например, выходной сигнал схемы рис. 1.1, *б* равен $AB+CD$. Это можно показать следующим образом:

$$g_4 = \overline{g_2 g_3} = \overline{g}_2 + \overline{g}_3.$$

Но

$$g_2 = \overline{A} + \overline{B}; \quad g_3 = \overline{C} + \overline{D}.$$

Тогда

$$g_4 = \overline{\overline{A} + \overline{B}} + \overline{\overline{C} + \overline{D}} = AB + CD.$$

Заинтересованный читатель подобным же образом может проанализировать выходные сигналы схем на рис. 1.1, *в* и 1.1, *г*.

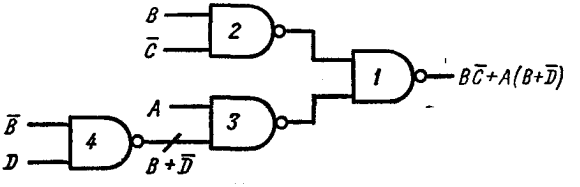


Рис. 1.2

Рассмотренные шаги, выполненные в обратном порядке, могут использоваться для реализации булевых функций с помощью элементов *И—НЕ*. Например, для реализации функции $f=\overline{BC}+A(B+\overline{D})$ поступаем следующим образом:

1. Изобразим выходной элемент *И—НЕ* — элемент *1* на рис. 1.2.
2. Затем добавим к нему два элемента *И-НЕ* по одному на каждое из произведений $B\overline{C}$ и $A(B+\overline{D})$.
3. Входные сигналы элемента *2* — B и $\overline{C}$, а элемента *3* — A и $B+\overline{D}$.
4. Формируем сигнал $B+\overline{D}$. Для этого подключаем элемент *4*, на вход которого подаются сигналы $\overline{B}$ и D, как показано на рис. 1.2.

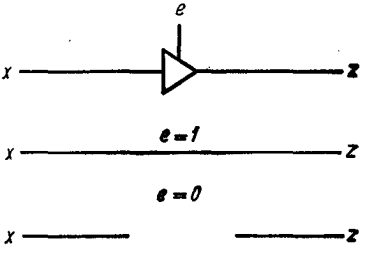


Рис. 1.3

Трехстабильные элементы были разработаны в 1969 г. Г. Майном (H. Mine) в университете г. Киото. Каждый такой элемент имеет один вход x, один выход z и один управляющий вход e (рис. 1.3). Когда $e=1$, элемент представляет собой прямое соединение входа с выходом, т. е. $z=x$. Когда $e=0$, цепь находится в третьем состоянии, т. е. ведет себя как разорванная.

Гонками называют нежелательные кратковременные сигналы, которые возникают в логических схемах при определенных соотношениях задержек элементов и величин входных сигналов. Схема, в которой возникают паразитные импульсы-всплески на выходе при изменении входного сигнала A от 1 до 0 при $B=C=1$,

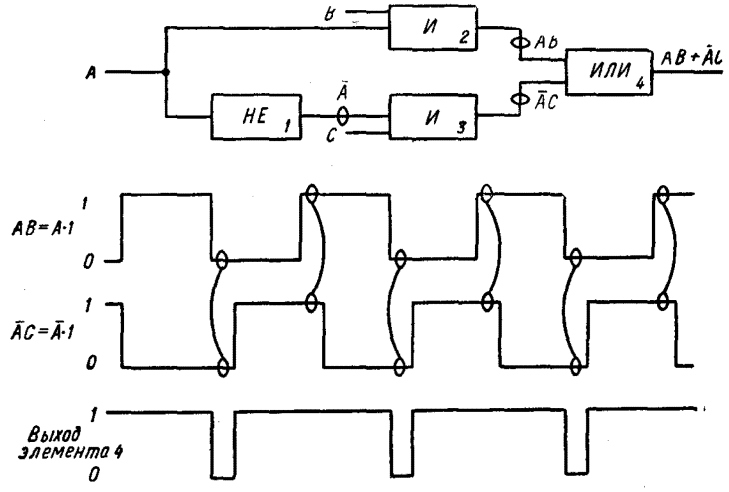


Рис. 1.4

показана на рис. 1.4. Причиной гонок являются непосредственные изменения сигнала A, $A=\bar{A}$, равные или 0, или 1. Это ведет к тому, что булево выражение для рассматриваемой схемы приводится к форме $A+\bar{A}$ или $A\cdot\bar{A}$, причем гонки существуют на выходе рассматриваемой схемы, несмотря на то, что входные сигналы свободны от гонок.

Возвратимся к примеру на рис. 1.4. Выражение $f=AB+\bar{A}C$ приводится к виду $A+\bar{A}$ при $B=C=1$, что указывает на существование гоночной ситуации на выходе элемента 4. Гонки в схеме легко устранить, если ее булево выражение лишить возможности приведения к одной из двух форм $A+\bar{A}$ или $A\cdot\bar{A}$. Это легко сделать с помощью теоремы 2:

$$AB + \bar{A}C = AB + \bar{A}C + BC$$

или

$$(A + B)(\bar{A} + C) = (A + B)(\bar{A} + C)(B + C).$$

Введение третьего члена препятствует приведению первичного выражения к виду $A+\bar{A}$ при $B=C=1$, так как при этом $AB+\bar{A}C+BC$ приводится к $A+\bar{A}+1=1$. Аналогично, когда $B=C=$

$=0$, второе выражение приводится к виду $(A+0)(\bar{A}+0)(0+0)=$ $=A\cdot\bar{A}\cdot 0=0$.

Для схем, спроектированных по вышеуказанному алгоритму и выполненных на элементах с максимальным разбросом задержки в $\pm 33^1/_3\%$, гонки автоматически исключаются.

Шаги алгоритма описаны в параграфе 1.9, а разброс задержек обоснован в 1.8.

1.6. НЕИСПОЛЬЗУЕМЫЕ СОСТОЯНИЯ

Если число используемых состояний логической схемы N, где $2^{n-1}<N<2^n$, то число неиспользуемых состояний равно $2^n—N$.

Читателю настоятельно рекомендуется не использовать такие состояния ввиду их неопределенности. Разработчик должен при проектировании предусмотреть условия и средства, исключающие попадание системы в неиспользуемые состояния. Другими словами, все диаграммы состояний содержат все 2^n состояний до этапа реализации схемы.

1.7. УМЕНЬШЕНИЕ ЧИСЛА СОСТОЯНИЙ

При определенных условиях можно уменьшить число состояний на диаграмме состояний. Эти условия были определены Калдвеллом (S. H. Caldwell) [4]. Рассмотрим эти условия.

Диаграмма состояний переводится в таблицу состояний (таблицу переходов), которая имеет столько строк, сколько состояний в системе, и столько столбцов, сколько имеется возможных комбинаций входных сигналов. Каждая строка соответствует состоянию на диаграмме, а каждый столбец — входному сигналу. Строки и столбцы отмечаются обозначениями состояний. Пересечение столбца и строки определяет состояние, в которое система перейдет из состояния, соответствующего данной строке, при подаче входного сигнала, соответствующего данному столбцу (для тактируемых схем тактовый сигнал не указывается в таблице состояний, однако подразумевается его обязательное присутствие для выполнения переходов). Если разработчик не может определить следующего состояния для данных конкретных условий, он не заполняет данную клетку. Для диаграмм состояний должны быть определены выходные сигналы для каждого состояния, кроме тех клеток, для которых состояния не определены. Очевидно, что если система переходит в состояние, совпадающее со своим предыдущим состоянием, то она устойчива по отношению к данной комбинации входных сигналов. Такое состояние отмечается на диаграмме жирной цифрой.

Процедура комбинирования строк в таблице состояний заключается в следующем.

1. Две строки могут быть слиты или объединены, если состояния системы и выходные сигналы в соответствующих столбцах

обеих строк одинаковы или соответствующие элементы одной или обеих строк пусты.

2. При объединении выделенного и невыделенного состояний с одним номером результирующее состояние будет выделенным.

Таким образом, две строки

	3	5	и		
	3	5		6	8
объединяются в	3	5		6	8.

Заметим, что изменение состояния системы из состояния 5 в состояние 8 теперь инициируется только входным сигналом. Когда строка s_m сливается со строкой s_n, новую строку необходимо именовать s_{mn}.

С применениями алгоритмов можно ознакомиться ниже. Дополнительные сведения приведены в [1].

1.8. ПОСЛЕДОВАТЕЛЬНОСТНЫЕ УРАВНЕНИЯ

Функционирование нетактируемых последовательностных схем алгебраически можно описать с помощью булевых выражений, обычно именуемых последовательностными уравнениями [1, 2]. Именно появление этих уравнений в 1969 г. дало возможность развить четкую и детальную процедуру реализации логических схем, в которой конструктивные ограничения принимаются во внимание на этапе проектирования.

Существует две основные формы последовательностных уравнений:

$$A = \sum \text{Уст. } A + A \overline{\sum \text{Сбр. } A}; \tag{1.1}$$

$$A = [\sum \text{Уст. } A + A] \overline{\sum \text{Сбр. } A}, \tag{1.2}$$

где переменная A соответствует вторичному сигналу (переменной состояния); Уст. A (условие установки вторичного сигнала) — совокупность (произведение) булевых переменных, при равенстве которого единице вторичный сигнал переводится во включенное состояние (т. е. в «1»); Сбр. A (условие сброса вторичного сигнала) — произведение булевых переменных, которое при равенстве единице вызывает переход вторичного сигнала в выключенное состояние (т. е. в «0»).

Будем называть такие уравнения элементарными последовательностными, и схемы, непосредственно их реализующие,— элементарными последовательностными.

Уравнение (1.1) используется при проектировании схем на элементах *И—НЕ*, а уравнение (1.2) — на элементах *ИЛИ—НЕ*. В дальнейшем будем называть эти уравнения соответственно *И—НЕ* и *ИЛИ—НЕ* последовательностными уравнениями.

Последовательностные уравнения можно реализовать не только на электронных элементах *И—НЕ, ИЛИ—НЕ*, но и на других типах цифровых элементов (электромеханических, гидравличе-

ских реле и т. д.). Схема реализации соответственно *ИЛИ—НЕ* и *И—НЕ* последовательностных уравнений на релейных элементах показана на рис. 1.5. Нажатие замыкающей кнопки s приводит к выполнению условий установки (включения), а нажатие размыкающей кнопки r — к выполнению условий выключения (сброса) реле A.

Если по некоторой причине условия установки и сброса существуют одновременно, то для уравнения *И—НЕ* условие установки превалирует над условием сброса. Для уравнения *ИЛИ—НЕ* справедливо обратное утверждение. Это свойство используют при проектировании помехоустойчивых схем. Условия установки и сброса вторичных сигналов можно получить непосредственно из диаграммы состояния. Например, согласно приведенной на рис. 1.6,*а* диаграмме двухтактного T триггера, можно записать:

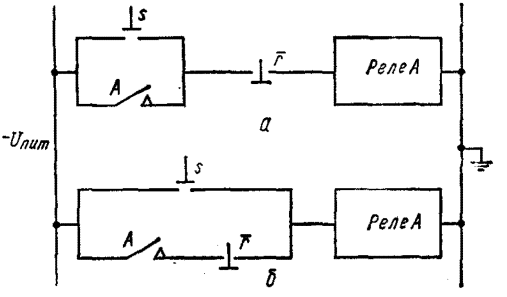


Рис. 1.5

условие установки $A = B\bar{c}$;
условие сброса $A = \bar{B}\bar{c}$;
условие установки $B = \bar{A}c$;
условие сброса $B = Ac$.

Подставляя эти величины в уравнение (1.1), получаем описание схемы *И—НЕ*:

$$A = B\bar{c} + A(B + c); \quad B = \bar{A}c + B(\bar{A} + \bar{c});$$
$$Z = S2 + S3 = AB + A\bar{B} = A.$$

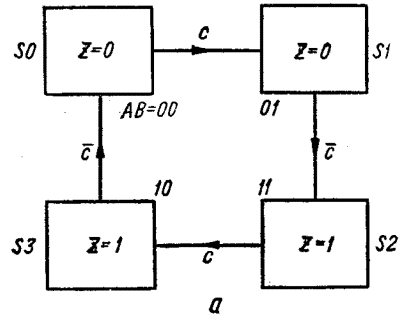
Схема, реализующая эти уравнения, показана на рис. 1.6, *б**. Число элементов в схеме можно уменьшить, применяя к исходным уравнениям процедуры слияния и замещения сигналов. Хотя эти процедуры и формализованы, они затрудняют проектирование и изменяют задержку сигналов. Поэтому они здесь не рассматриваются. Заинтересованный читатель может найти их описание в работе [2].

Инверсные сигналы. Докажем, что на выходе элементов *4* и *8* (рис. 1.6, *б*) формируются сигналы $\bar{A}$ и $\bar{B}$ соответственно. Обозначим через s сумму условий установки вторичного сигнала M,

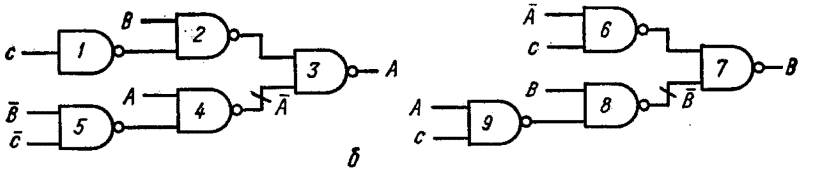
* Реализация получена из уравнений, преобразованных к виду $A = B\bar{c}A\bar{B}c$; $B = \bar{A}cB\bar{A}c$.

а через r — сумму условий сброса M, тогда $M=s+M\bar{r}$. Схема реализации этого уравнения с помощью элементов *И—НЕ* показана на рис. 1.7. Для нахождения сигнала $\bar{M}$ проинвертируем обе части уравнения $\bar{M}=\bar{s}(\bar{M}+r)$. Добавим $s\bar{M}$ и sr как дополнительные члены к правой части уравнения. Получим

$$\begin{aligned}\bar{M} &= \bar{s}\bar{M} + \bar{s}\bar{r} + (s\bar{M}) + (sr)\\ &= \bar{M}(\bar{s}+s) + r(\bar{s}+s)\\ &= \bar{M} + r\\ &= \text{выход элемента } 3 \text{ на рис. 1.7.}\end{aligned}$$



а



б

Рис. 1.6

Произведение $s\bar{M}$ можно использовать как дополнительное, так как при $s=1$ M не равно нулю. Аналогично s и r никогда не могут быть равны единице одновременно, являясь условием установки и сброса вторичного сигнала.

Задержки сигналов элементами [5] логических схем можно определить по схеме или непосредственно из булевого уравнения схемы, как показано ниже. Пусть t_g — номинальное время задержки элемента. Для схемы из рис. 1.6, *б* можно записать:

время установки $A = 3t_g$ — элементы *1*, *2* и *3*;
время сброса $A = 4t_g$ — элементы *1*, *5*, *4* и *3*;
время установки $B = 2t_g$ — элементы *6* и *7*;
время сброса $B = 3t_g$ — элементы *9*, *8* и *7*.

Определим время задержки по последовательностным уравнениям схемы на примере уравнений: $A=B\bar{c}+A[B+c]$; $B=\bar{A}c+$ $+B[\bar{A}+\bar{c}]$.

Сигнал A устанавливается, когда условие его установки $B\bar{c}=1$, т. е. когда $B=1$ и $\bar{c}=1$. Временной интервал между из-

менением c и изменением A равен $3t_g s$, так как сигнал c сначала инвертируется, затем объединяется по схеме *И* с сигналом B и, наконец, объединяется по схеме *ИЛИ* с сигналом $A[B+c]$ прежде, чем вызвать установку A в 1. Аналогично

время сброса $A = 4t_g$ — схемы *НЕ, ИЛИ, И, ИЛИ*;
время установки $B = 2t_g$ — *И, ИЛИ*;
время сброса $B = 3t_g$ — *ИЛИ, И, ИЛИ*.

Выбранная форма последовательностных уравнений определяет прохождение измененного входного сигнала, по крайней мере, через один уровень элементов *И* и через один уровень элементов *ИЛИ*, перед тем как изменится вторичный сигнал. Таким образом, кратчайшее время, за которое изменится вторичный сигнал (переменная состояния), составляет $2t_g$.

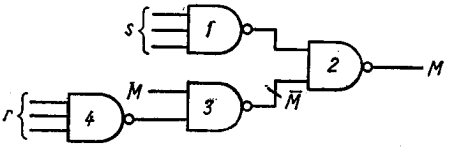


Рис. 1.7

Максимальное время задержки входного сигнала при инвертировании равно t_g. Обозначим через $x\%$ максимальный разброс значения задержки элемента, обусловленный такими факторами, как технология, различия в нагрузке, старение и т. д., тогда получим

$$t_{p\,\max} = t_g(1 + x);\ t_{s\,\min} = 2t_g(1 - x).$$

Как показано в работах [1, 2, 3], задержки в схемах несущественны, если $t_{p\,\max} \leqslant t_{s\,\min}$.

Таким образом, для элементарных схем

$$t_g(1 + x) \leqslant 2t_g(1 - x);$$
$$1 + x \leqslant 2 - 2x;$$
$$x \leqslant 1/3.$$

Следовательно, схемы свободны от гонок, если они построены на элементах с максимальным разбросом задержек $\pm 33^1/_3\%$. Эта цифра — теоретический предел. Практически ее можно увеличить путем включения наиболее вероятных медленных элементов в критический путь. Дальнейшее увеличение может быть достигнуто, если принять во внимание фильтрующие свойства элементов.

Автор обращает внимание читателя на необходимость избегать алгебраических преобразований последовательностных уравнений, учитывая, что каждое алгебраическое преобразование изменяет относительные задержки первичных и вторичных сигналов. Для минимизации схемы разработчик должен применять процедуру слияния и подстановки сигналов [2].

1.9. АСИНХРОННЫЕ НЕТАКТИРУЕМЫЕ ПОСЛЕДОВАТЕЛЬНОСТНЫЕ СХЕМЫ

Ниже описано проектирование асинхронных последовательностных схем.

Особенности проектирования. Описываемый процесс проектирования состоит из четырех шагов. Он должен удовлетворять следующим требованиям.

1. *Надежность схемы.* Все функции, описывающие схему, корректны и правильны.
2. *Минимальность схемы.* Часто не все получаемые схемы минимальны.
3. *Разброс задержек элементов.* Подразумевается $\pm 33^1/_3\%$ разброс задержек.
4. *Простота обслуживания.* Проектируемые схемы просты в обслуживании.
5. *Затраты времени на проектирование.* Минимальны.
6. *Документирование.* Никакие дополнительные документы не требуются.
7. *Алгоритм проектирования.* Прост в применении. Не требуются никакие специальные знания в электронике.
8. *Коэффициент разветвления по входу и выходу.* Удовлетворяется, но не столь эффективно, как читатель может найти в работе [2].

Алгоритм проектирования. Ниже детально описаны шаги проектирования и их последовательность.

Шаг 1 — характеристики входа и выхода. На этом шаге составляют диаграмму состояний, на которой обозначают допустимые входные и требуемые выходные сигналы. Далее диаграмму состояний используют для определения отношений между входными и выходными сигналами, реализуемых схемой.

Шаг 2 — внутренние состояния. На 2-м шаге разработчик определяет внутреннюю структуру схемы. Наряду с опытом, интуицией и предвидением, играющими важную роль на этом этапе, для неопытного разработчика важна уверенность в полноте и однозначности решения. По этой причине он должен избегать сокращения алгоритма и использовать столько состояний, сколько он считает необходимым для полного и однозначного описания функционирования схемы. Нежелательные состояния можно удалить на следующем шаге.

Шаг 3 — уменьшение числа состояний. Этот шаг необязателен и может быть опущен. Главная его цель — уменьшить число внутренних состояний, полученных на 2-м шаге, если такое уменьшение возможно и целесообразно.

В таблице состояний схемы, используя алгоритм, уменьшается число состояний путем слияния строк. Для отбрасывания избыточных состояний необходимо уменьшать число состояний лишь до некоторой степени двойки. Например, можно исполь-

зовать этот шаг для уменьшения числа состояний от пяти до четырех, но нельзя использовать его для уменьшения от четырех состояний до трех.

Шаг 4 — реализация цепи. На этом шаге находится однозначное соответствие между внутренними состояниями и двоичным кодом. Кодирование состояний должно быть таким, чтобы переход из одного состояния в другое сопровождался изменением только одного двоичного разряда в этом коде. Для этой цели можно использовать свободную от гонок диаграмму (рис. 1.8).

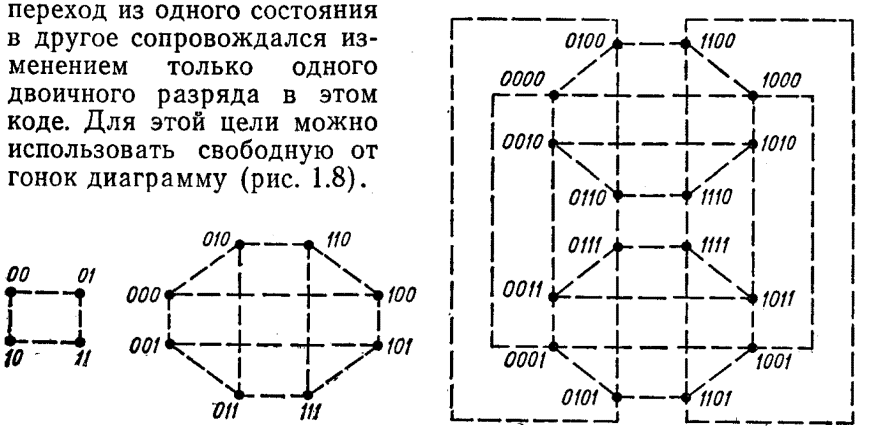


Рис. 1.8

После кодирования внутренних состояний осуществляются преобразования булевых уравнений для переменных состояний

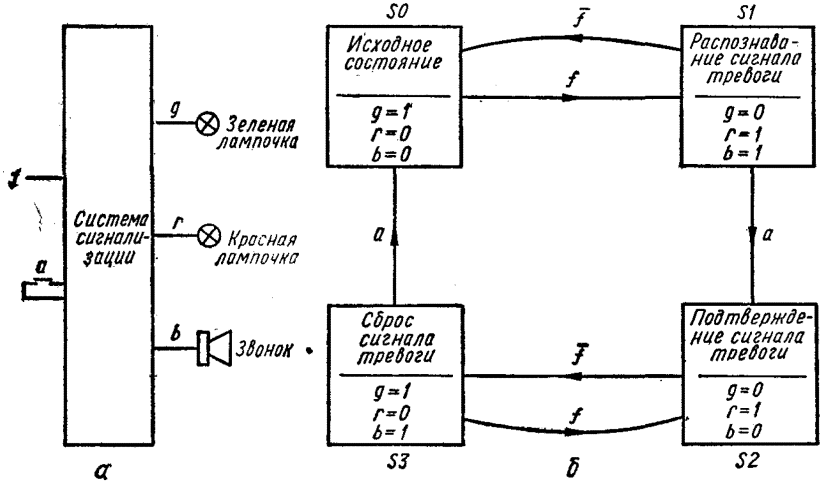


Рис. 1.9

и выходных сигналов. Хотя первоначально в таблице состояний могут существовать пустые клетки, после преобразования

уравнений схемы они должны быть заполнены. Пустые места заполняются дополнительными членами, описывающими неопределенные состояния схемы. Другими словами, в таблице состояний для окончательной реализации схемы не должно быть пустых мест.

Пример 12. Разработать систему сигнализации. Спроектируем систему сигнализации со следующими характеристиками. По сигналу тревоги включается звонок, выключается зеленая лампочка и включается красная. Звонок может быть выключен оператором нажатием кнопки подтверждения. Если сигнал тревоги выключается сам, красная лампочка гаснет, зеленая загорается, а звонок автоматически включается для привлечения внимания оператора. Звонок выключается нажатием кнопки подтверждения. Если сигнал тревоги исчезает до нажатия кнопки подтверждения, система возвращается в исходное состояние [2].

af	00	01	11	10
S0	(S0) g,r,b=1,0,0	S1 ∅,∅,∅		(S0) 1,0,0
S1	S0 ∅,∅,∅	(S1) 0,1,1	S2 ∅,∅,∅	
S2	S3 ∅,∅,∅	(S2) 0,1,0	(S2) 0,1,0	
S3	(S3) 1,0,1	S2 ∅,∅,∅		S0 ∅,∅,∅

а

af	00	01	11	10
S01	(S01) 1 g,r,b=1,0,0	(S01) 2 0,1,1	S23 3 ∅,∅,∅=0,1,1	(S01) 4 1,0,0
S23	(S23) 5 1,0,1	(S23) 6 0,1,0	(S23) 7 0,1,0	S01 8 ∅,∅,∅=1,0,1

б

Рис. 1.10

Решение.

Шаг 1 — характеристики входа и выхода. Входные и выходные сигналы системы показаны на ее структурной схеме (рис. 1.9, *а*), а их соотношения — на диаграмме состояний (рис. 1.9, *б*).

Шаг 2 — внутренние состояния. В рассматриваемом случае внутренние характеристики совпадают с внешними.

Шаг 3 — уменьшение числа состояний. Составим таблицу состояний (рис. 1.10,*а*) по диаграмме состояний (рис. 1.9,*б*). Выходами схемы являются две лампочки и звонок, которые не реагируют на кратковременные импульсы. Обозначим их состояния как 0 или 1 на все время переходов в схеме из одного необведенного состояния в другое. Далее используем методику уменьшения числа состояний, описанную в параграфе 1.8. Строки *S0, S1* и *S2, S3* сливаются в строки *S01* и *S23,* соответственно сокращая число строк в таблице с четырех до двух (рис. 1. 10, *б*). Соответствующая диаграмма состояний показана на рис. 1.11, *а*.

Используя входы $\varnothing$ в нашей таблице с двумя состояниями как дополнительные члены, запишем:

в состоянии *S01* (рис. 1.10,*б*)

$g = \bar{a}\bar{f} + a\bar{f} + (af) = \bar{f}$, что соответствует $g = 0$ в ячейке с номером 3;
$r = \bar{a}\bar{f} + (af) = f$, что соответствует $r = 1$ в ячейке с номером 3;
$b = \bar{a}f + (af) = f$, что соответствует $b = 1$, в ячейке с номером 3;

в состоянии *S23*

$g = \bar{a}\bar{f} + (a\bar{f}) = \bar{f}$, что соответствует $g = 1$ в ячейке с номером 8;
$r = \bar{a}\bar{f} + af + (a\bar{f}) = f$, что соответствует $r = 0$ в ячейке с номером 8;
$b = \bar{a}\bar{f} + (a\bar{f}) = \bar{f}$, что соответствует $b = 1$ в ячейке с номером 8.

Шаг 4 — реализация цепи. Непосредственно из рис. 1.11, *а* получаем: условие установки $A = af$; условие сброса $A = \overline{af}$.

Таким образом, $A = af + A\overline{(a + f)}$;

$$g = S01\overline{f} + S23\overline{f} = \overline{A}\overline{f} + A\overline{f} = \overline{f};$$
$$r = S01f + S23f = \overline{A}f + Af = f;$$
$$b = S01f + S23\overline{f} = \overline{A}f + A\overline{f}.$$

Соответствующая *И-НЕ* схема показана на рис. 1.11, *б*.

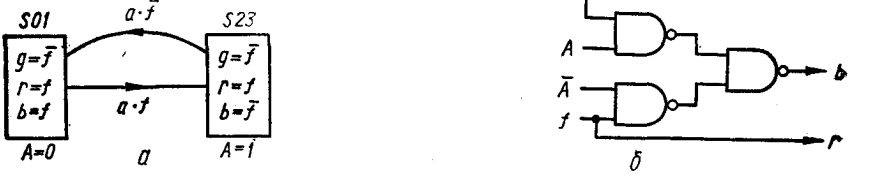


Рис. 1.11

1.10. СИНХРОННЫЕ ТАКТИРУЕМЫЕ ПОСЛЕДОВАТЕЛЬНОСТНЫЕ СХЕМЫ

Синхронные схемы характеризуются следующими особенностями:

1) работа таких схем синхронизирована тактовыми импульсами, при отсутствии которых состояние схемы неизменно;

2) любое число переменных состояния может измениться во время переключений схемы.

На этапе схемной реализации переменная состояния представляет собой двухтактный триггер, т. е. бистабильный элемент, в котором изменение выходного сигнала совпадает по времени с фронтом или спадом тактового импульса. Предполагается, что изменение выходного сигнала A происходит синхронно со спадом тактового импульса. Другие случаи будут рассмотрены особо.

Существует четыре основных типа триггеров:

1) D-триггер
2) T-триггер
3) RS-триггер
4) JK-триггер.

Их обозначения и диаграммы состояний показаны на рис. 1.12. Методы их реализации рассмотрены в работе [1].

Проектирование синхронных схем выполняется по алгоритму, состоящему из четырех шагов. Алгоритмы проектирования синхронных схем отличаются от алгоритма проектирования асинхронных схем. Переменные состояния определяются уравнениями

триггеров в отличие от последовательностных уравнений для асинхронных схем. Уравнения триггеров представляют собой булевы выражения условий установки и сброса триггеров. Условие установки RS-триггера, обозначаем как S_A, является объединением *(ИЛИ)* общих состояний*, вызывающих изменение величины A от 0 к 1. Аналогично условие сброса A, обозначаемое как R_A, представляет собой дизъюнкцию общих состояний, каждое из которых вызывает сброс величины A от 1 до 0.

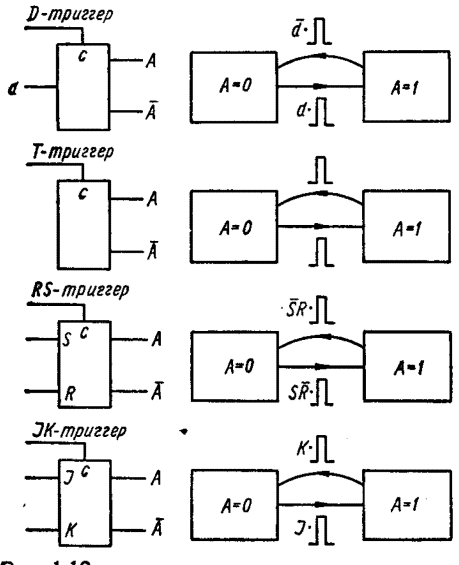


Рис. 1.12

Выражения для условий установки и сброса триггеров можно минимизировать, используя дополнительные члены: а) определяющие нерабочие комбинации сигналов; б) определяющие появление условия установки триггера в процессе переключения, когда выход триггера находится в состоянии 1; в) определяющие появление условия сброса триггеров во время нахождения выхода триггера в состоянии 0.

Условия установки и сброса в таком виде, как они определены выше, представляют собой непосредственно сигналы уста-

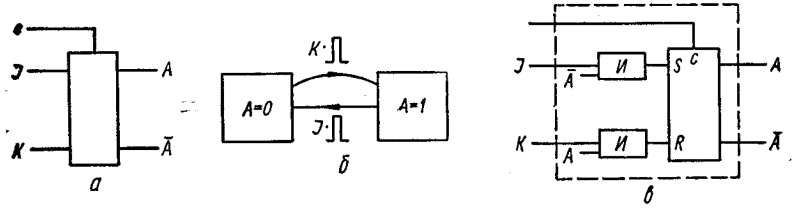


Рис. 1.13

новки и сброса для RS-триггера. Однако на практике чаще используются универсальные JK-триггеры. Для получения уравнений сигналов J и K выделим переменные $\bar{A}$ и A из уравнений, описывающих S_A и R_A. Наиболее легкий метод выполнения такого преобразования — прямая реализация JK-триггера с по-

* Общими являются состояния, определяемые неповторяющейся комбинацией входных и вторичных сигналов.

мощью RS-триггера. Графическое изображение и диаграмма состояний JK-триггера показаны на рис. 1.13. Внутренние параметры этого триггера не отличаются от внешних. Таким образом, непосредственно из рис. 1.13, *б* получим: $S_A = S0J = \bar{A}J$; $R_A = S1K = AK$.

Соответствующая схема JK-триггера показана на рис. 1.13, *в*.

Рассмотрим алгоритм проектирования синхронной схемы на следующем примере (более подробно см. [2]).

Пример 13. Детектор 4—5—6. Спроектировать схему, которая будет останавливать считыватель перфоленты путем выключения сигнала *m* и включать звонок, если встречается последовательность символов *4—5—6* (рис. 1.14).

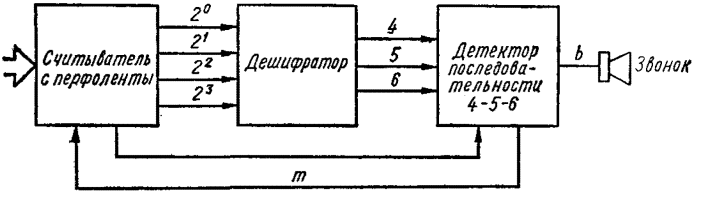


Рис. 1.14

Импульс синхронизации вырабатывается считывателем при считывании каждого нового символа.

Решение.

Шаг 1 — характеристика входа и выхода. Определены в условии.

Шаг 2 — внутренние состояния. Соответствующая внутренняя диаграмма состояний показана на рис. 1.15,*а*. В первоначальном состоянии *S0* детектор ждет появления символа *«4»*, игнорируя все остальные символы. При обнаружении символа *«4»* детектор переходит в состояние *S1*. В состоянии *S1* схема ожидает второй символ искомой последовательности *«5»*. При появлении символа *«5»* система переводится в состояние *S2*. Появление символа *«4»* в состоянии *S1* не вызывает выхода из этого состояния, поскольку искомой последовательности *4—5—6* может предшествовать сколько угодно четверок. Появление символа, отличного от *«4»* или *«5»*, т. е. $\bar{4}\cdot\bar{5}$, вызывает переход системы в первоначальное состояние *(S0)*.

В состоянии *S2* система ожидает символ *«6»*, появление которого переводит его в состояние *S3*, выключает считыватель и включает звонок. Появление символа *«4»* вызывает переход в состояние *S1*, а все остальные символы $(\bar{4}\cdot\bar{6})$ переводят схему в состояние *S0*.

Шаг 3 — минимизация числа состояний. Соответствующая таблица состояний показана на рис. 1.15,*б*. Как видно, слияние строк невозможно.

Шаг 4 — построение схемы. Для нумерации четырех состояний выберем произвольные коды. В процедуре реализации не будем использовать дополнительные члены для минимизации схемы. Непосредственно из диаграммы состояний (рис. 1.15,*а*) следует

$$S_A = S1 \cdot 5 = \bar{A}B \cdot 5, \text{ поэтому } J_A = B \cdot 5;$$

$$R_A = S2 \cdot 4 + S2 \cdot \bar{4} \cdot \bar{6} = S2 \cdot 4 + S2 \cdot \bar{6} =$$

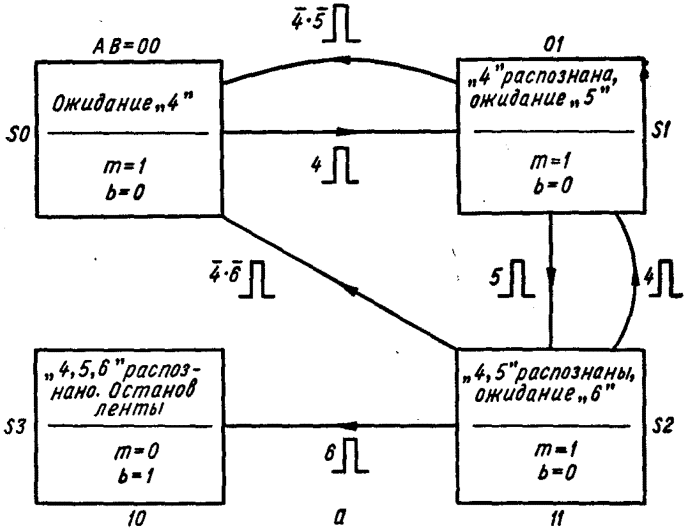
$$= S2 \cdot \bar{6}, \text{ так как } S2 \cdot 4 \text{ является подмножеством } S2 \cdot \bar{6};$$

$$= AB \cdot \bar{6}, \text{ поэтому } K_A = B \cdot \bar{6};$$

$$S_B = S0 \cdot 4;$$

$$= \bar{A}\bar{B} \cdot 4, \text{ поэтому } J_B = \bar{A} \cdot 4;$$

$$R_B = S1 \cdot \overline{4} \cdot \overline{5} + S2 \cdot \overline{4} \cdot \overline{6} + S2 \cdot 6 =$$
$$= S1 \cdot \overline{4} \cdot \overline{5} + S2 \cdot \overline{4} + S2 \cdot 6 =$$
$$= S1 \cdot \overline{4} \cdot \overline{5} + S2 \cdot \overline{4}, \text{ поскольку } S2 \cdot 6, \text{ это подмножество } S2 \cdot \overline{4}$$
$$= \overline{AB} \cdot \overline{4} \cdot \overline{5} + AB \cdot \overline{4}$$
$$= B \cdot \overline{4} \cdot \overline{5} + AB \cdot \overline{4}, \text{ поэтому } K_B = \overline{4} \cdot 5 + A \cdot \overline{4};$$
$$m = \overline{S3} = \overline{A} + B,$$
$$b = S3 = A\overline{B}.$$



б

Вход	4	5	6	$\overline{4}\cdot\overline{5}\cdot\overline{6}$
S0	S1 m=1 b=0	(S0) m=1 b=0	(S0) m=1 b=0	(S0) m=1 b=0
S1	(S1) m=1 b=0	S2 m=1 b=0	S0 m=1 b=0	S0 m=1 b=0
S2	S1 m=1 b=0	S0 m=1 b=0	S3 m=1 b=0	S0 m=1 b=0
S3			m=0 b=1	

в

Вход	4	5	6	$\overline{4}\cdot\overline{5}\cdot\overline{6}$
S0	S1 m=1 b=0	(S0) m=1 b=0	(S0) m=1 b=0	(S0) m=1 b=0
S1	(S1) m=1 b=0	S2 m=1 b=0	S0 m=1 b=0	S0 m=1 b=0
S2	S1 m=1 b=0	S0 m=1 b=0	S3 m=1 b=0	S0 m=1 b=0
S3	(S3) m=0 b=1	(S3) m=0 b=1	(S3) m=0 b=1	(S3) m=0 b=1

Рис. 1.15

Перед реализацией уравнений необходимо заполнить все пустые клетки таблицы состояний с тем, чтобы реакция схемы была определена для всех входных сигналов. Наиболее просто это можно сделать, опираясь на уравнение схемы, как описано ниже.

В таблице рис. 1.15,*б* есть четыре пустые клетки, их содержимое может быть описано булевым выражением $A \cdot \overline{B}$, т. е. $A = 1$, $B = 0$.

Подставляя эти величины в триггерные уравнения, получаем:

$$J_A = B \cdot 5 = 0;$$
$$K_A = B \cdot \overline{6} = 0;$$
$$J_B = \overline{A} \cdot 4 = 0;$$
$$K_B = 4 \cdot \overline{5} + A \cdot \overline{4} = \overline{4} \cdot \overline{5} + \overline{4} = \overline{4}.$$

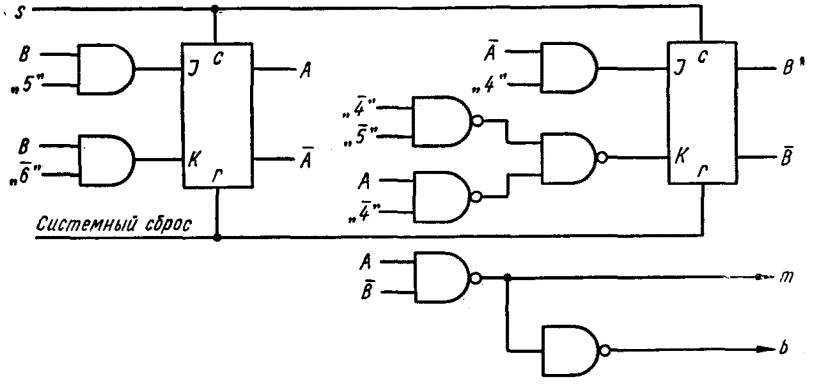


Рис. 1.16

Это показывает, что переход системы в состояние *S3* (выключение двигателя) блокирует JK-триггер A в установленном состоянии ($J_A = K_A = 0$) и JK-триггер B в сброшенном состоянии (так как $J_B = 0$). Другими словами, состояние *S3* вводится в пустые клетки таблицы состояний (рис. 1.15, *а*). Соответствующая схема показана на рис. 1.16.

1.11. СПИСОК ЛИТЕРАТУРЫ

1. **Zissos D.** Problems and Solutions in Logic Design. Oxford University Press, 1976.
2. **Zissos D.** Logic Design Algorithms. Oxford University Press, 1972.
3. **Zissos D.** Race-hazards. — Process Control by Power Fluidics. Proceedings of an International Symposium of the Institute of Measurment. Sheffield. U. K., September, 1975.
4. **Caldwell S. H.** Switching Circuits and Logic Design. Wiley, 1965.
5. **Duncan F. G., Zissos D.** Gate Tolerance in Sequential Circuits. — Proc. IEE, 118, № 2, February, 1971.
6. **Duncan F. G., Zissos D.** Design of Synchronous Multi-level Sequential Circuit. — Proc. IEE, 119, № 2, February, 1972.

ГЛАВА 2

МИКРОПРОЦЕССОРЫ

В этой главе рассматриваются общие характеристики микропроцессорных схем, причем особое внимание уделяется характеристикам, существенным при проектировании. Уточняется методика проектирования, изложенная в предыдущей главе.

2.1. ОПИСАНИЕ МИКРОПРОЦЕССОРА

Определение. Микропроцессор — это программно управляемая синхронно-последовательностная система * (см. параграф 1.10), которая способна выполнять следующие функции: 1) вы-

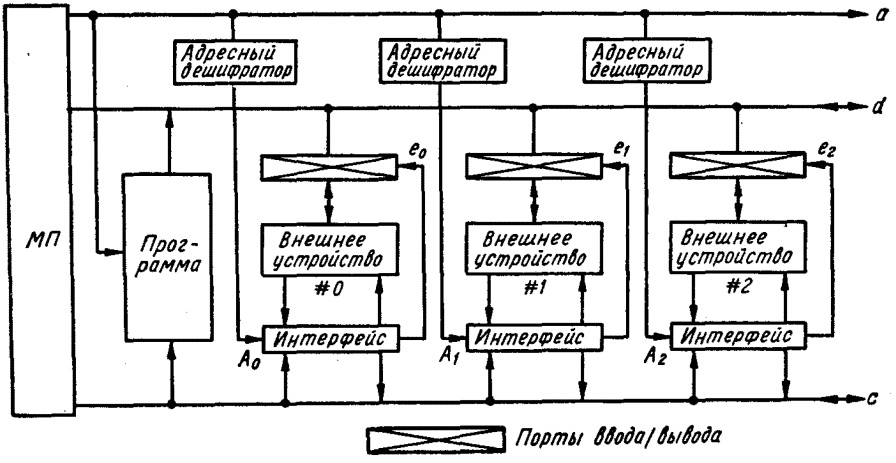


Рис. 2.1

полнять программу, т. е. последовательность команд, используемую для реализации поставленной задачи; 2) управлять системой с шинной организацией (рис. 2.1, функционирование и назначение компонентов этой системы описаны ниже).

Программа обычно хранится в полупроводниковой памяти — ОЗУ, ПЗУ или ППЗУ. Особенности этих типов ЗУ рассмотрены ниже.

---

* Синхронно-последовательностная система — это логическая схема со многими состояниями, функционирование которой синхронизируется тактовыми импульсами, а в промежутках между импульсами состояние системы не изменяется.

Прежде чем перейти к детальному ознакомлению с микропроцессором (МП), необходимо отметить, что его внутренняя структура по своему составу и архитектуре ничем не отличается от современных ЭВМ. В частности, МП содержит регистры, арифметико-логическое устройство (АЛУ), дешифраторы, флажки состояний и т. д. Основное отличие МП — быстровозрастающая плотность размещения компонентов.

Увеличение плотности — одна из наиболее важных проблем в технологии БИС. Дело в том, что физические ограничения на число выводов (14, 16, 24 и 40 выводов стандартного корпуса микросхемы) вступают в противоречие с возрастающим числом логических элементов на кристалле. Для МП противоречие это

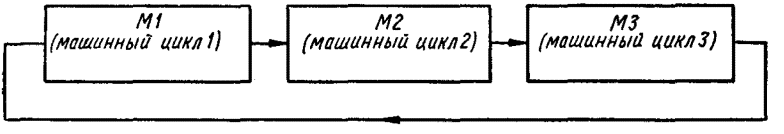


Рис. 2.2

устранено применением метода разделения времени для входных и выходных сигналов. При этом один и тот же вывод микросхемы используется, например, для ввода данных и команд в МП и для вывода данных из него. Процесс разделения времени на выводах МП рассмотрен ниже.

Разделение во времени. Для упрощения изложения компоненты схемы, не имеющие отношения к разделению времени в МП, опущены. Упрощенная структурная схема МП показана на рис. 2.3, где изображены лишь четыре основных регистра: программный счетчик *PC*, содержащий адрес следующей команды; аккумулятор *AC*, предназначенный для хранения принимаемых и выдаваемых микропроцессором данных; адресный регистр *r*, содержимое которого используется для адресации памяти при чтении и записи; регистр кода операции *IR*, принимающий код операции и хранящий его в течение времени выполнения команды.

Указанные регистры представляют собой минимальный набор регистров, необходимых МП. Обычно МП содержит и другие регистры, например указатель стека, индексный регистр и т. д. Функции этих регистров и внутренняя организация МП рассмотрены в работе [1].

Рассматриваемый простой микропроцессор выполняет команду ввода/вывода за три машинных цикла: *M1, M2* и *M3* (рис. 2.2) (машинный цикл *M2* может быть выполнен дважды, если, например, обмен МП с памятью производится байтами, а адрес имеет длину два байта). Машинный цикл *M1* имеет четыре последовательных состояния, машинные циклы *M2* и *M3* — по три состояния (рис. 2.4).

Предположим, что программный счетчик *PC* загружен адресом следующей команды, которая представляет собой команду

ввода/вывода, процесс выполнения которой описан ниже (при изучении процесса выполнения команды читателю рекомендуется обратить внимание на рис. 2.3 и 2.4).

Машинный цикл M1. В течение этого машинного цикла, называемого циклом извлечения команды, из памяти извлекается команда, подлежащая выполнению, и пересылается в регистр кода операции.

Если обозначить через A_m ячейку памяти, содержащую следующую команду, то содержимое программного счетчика *PC*

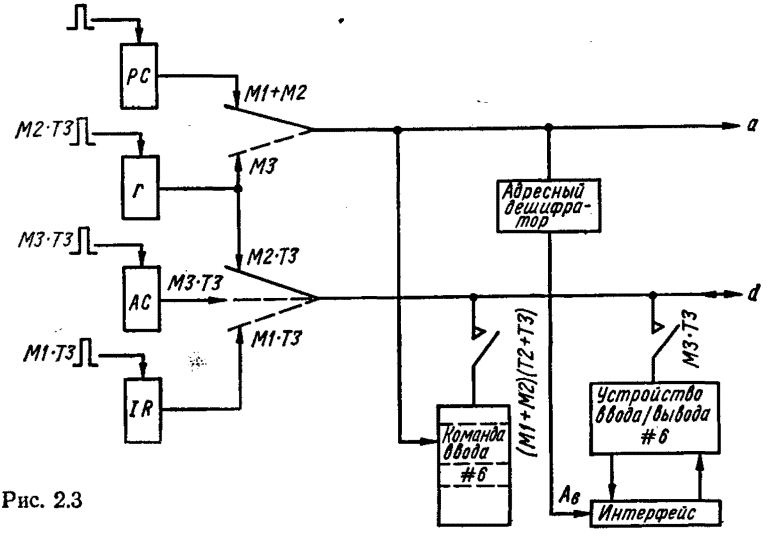


Рис. 2.3

перед выполнением этого цикла равно A_m. Когда МП переходит в состояние *M1·T1* (рис. 2.4), адресные шины подсоединяются через внутренний двунаправленный формирователь к *PC*, что позволяет выдать адрес ячейки A_m. Далее генерируется последовательность сигналов для извлечения содержимого ячейки памяти A_m и выдачи его на выходные контакты микросхемы памяти. В течение этого машинного цикла шина данных (*d*) не несет информации и находится в третьем состоянии.

Следующий синхросигнал (импульс тактовой частоты) переводит МП в состояние *M1·T2*. В этом состоянии ничего не изменяется: программный счетчик *PC* все так же присоединен к адресной шине (*a*), а шина *d* остается в третьем состоянии. Это состояние, как мы увидим ниже, соответствует введению задержки перед состоянием *M1·T3*, чтобы память успела выдать следующую команду. В связи с тем, что принципы синхронизации с памятью и внешними устройствами детально рассматриваются в следующем параграфе, будем пока считать, что внешние устройства и память обладают достаточно малым временем для обмена и не замедляют работу МП.

При входе микропроцессора в состояние *M1·T3* шина данных *d* подключается к регистру кода операции *IR* через внутренние трехстабильные буферы МП. Одновременно выход микросхемы памяти, в которой записана программа, подсоединяется к той же шине и таким образом осуществляется прямая связь между памятью и регистром кода операции, как показано на рис. 2.3. Это

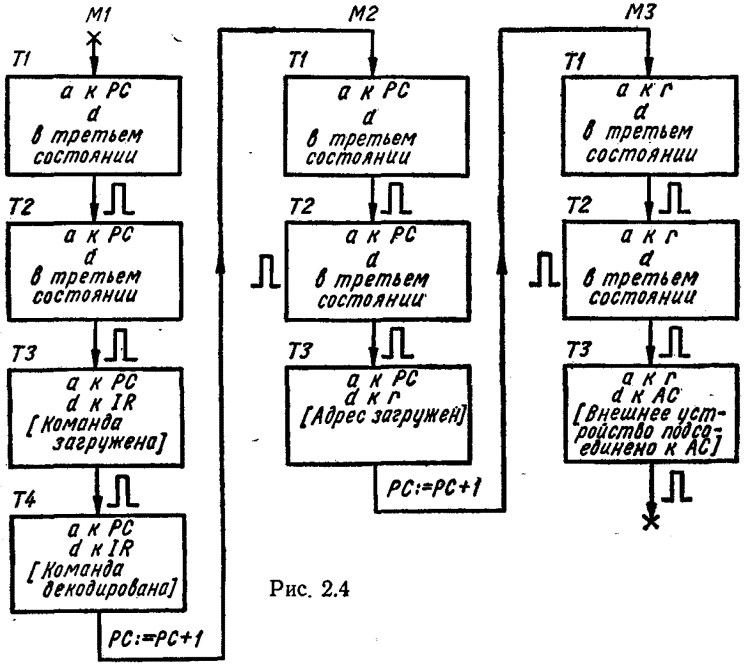


Рис. 2.4

означает, что на входе регистра *IR* находится следующая команда. Импульс, формируемый МП, передает эту команду в регистр кода операции.

Следующий синхроимпульс переводит МП в состояние *M1·T4* (рис. 2.4). В этом состоянии содержимое регистра *IR*, т. е. выполняемая команда, декодируется, а содержимое программного счетчика *PC* увеличивается на 1. Четвертый синхроимпульс переводит МП в машинный цикл *M2*.

Машинный цикл M2. Этот цикл называют также циклом извлечения адреса, так как этот цикл используется для извлечения из памяти и засылки в МП адреса внешнего устройства. Механизм пересылки адреса идентичен механизму пересылки команды из памяти в МП в предыдущем цикле. В течение этого цикла программный счетчик *PC* подключен к адресной шине *a*.

В состоянии *M2·T1* сигнал шины *a* извлекает содержимое следующей ячейки памяти. Как и в состоянии *M1·T1*, шина *d* находится в третьем состоянии.

В состоянии *M2·T2*, как и в состоянии *M1·T2*, ничего не изменяется. В этом состоянии осуществляется синхронизация с памятью.

Когда микропроцессор переходит в состояние *M2·T3*, шина данных *d* подсоединяется к адресному регистру *r*. Одновременно выход микросхемы памяти, содержащей программу, также подсоединяется к шине данных, устанавливая таким образом прямую связь между памятью и адресным регистром (рис. 2.3). Это означает, что на вход адресного регистра подается номер устройства ввода/вывода. Формируемый МП импульс заносит этот номер в адресный регистр. Седьмой синхроимпульс увеличивает содержимое *PC* на 1 и переводит систему в состояние *M3* (рис. 2.4).

Машинный цикл M3. Этот цикл называют циклом выполнения ввода/вывода, так как в течение этого цикла устанавливается прямая связь между МП и внешним устройством, определяемым номером, извлеченным в цикле *M2*.

В течение машинного цикла *M3* адресная шина *a* подсоединяется к адресному регистру *r*, обеспечивая выдачу номера внешнего устройства в течение всего цикла *M3*.

В состоянии *M3·T1* и *M3·T2*, как и в соответствующих состояниях двух предыдущих циклов, шина данных *d* находится в третьем состоянии. В состоянии *M3·T1* в систему из МП выдается номер внешнего устройства, а состояние *M3·T2* используется для задержки перед переключением МП в состояние *M3·T3* (рис. 2.4). Эта задержка используется для синхронизации во времени, так как МП не должен перейти в состояние *M3·T3* до тех пор, пока адресуемое внешнее устройство не известит о своей готовности. Механизм такой задержки рассмотрен в параграфе 2.2. Для простоты предположим, что синхронизация не требуется.

В состоянии *M3·T3* шина данных подсоединяется к аккумулятору через двунаправленный формирователь МП. В то же время информационный выход внешнего устройства подсоединяется к шине данных, образуя прямой канал данных между внешним устройством и аккумулятором (рис. 2.3). В течение этого времени данные передаются по шине данных в направлении, определяемом командой (ввод или вывод).

2.2. СОСТОЯНИЕ ОЖИДАНИЯ

Синхронизация с памятью. Как уже указывалось, функционирование микропроцессора определяется синхроимпульсами (рис. 2.4). Чем выше частота синхроимпульсов, тем больше производительность системы. Максимальная частота синхронизации данной системы определяется задержками внутренних схем МП и временем доступа к памяти программ. Существующие в настоящее время микропроцессоры более быстродействующи, чем микросхемы памяти. Поэтому важно, чтобы разработчики систем

с МП понимали механизм синхронизации с памятью для использования МП на предельных частотах. Этот механизм описан ниже.

В течение цикла извлечения команды или адреса, т. е. в течение цикла обращения к памяти, частота МП должна быть достаточно низкой для того, чтобы дать микросхеме памяти возможность обработать запрос. В нашем случае (см. рис. 1.4) МП посылает адрес чтения памяти в состоянии *M1·T1*. В состоянии *M1·T3*, т. е. на два периода позже частоты синхронизации, выходные данные с выбранной микросхемы памяти передаются в регистр кода операции *IR* (рис. 2.3). Очевидно, что система не должна войти в состояние *M1·T3* до тех пор, пока память не ответила на запрос. Таким образом, максимальная тактовая частота f_{max} должна выбираться так, чтобы два ее периода были не меньше времени доступа к памяти t, т. е.

$$2/f_{max} \geqslant t, \qquad f_{max} \leqslant 2/t.$$

Отметим, что на практике разработчик не встречается с проблемой синхронизации с памятью, если МП и элементы памяти согласованы по максимальной тактовой частоте. Если требуется реализовать память на элементах, не согласованных по максимальной тактовой частоте, можно использовать методы синхронизации, применяемые при обмене с внешними устройствами.

Синхронизации ввода/вывода. Время цикла ввода/вывода МП должно быть согласовано с временем ответа внешнего устройства. Например, если во время операции ввода/вывода необходимо продвинуть перфоленту на одну позицию для чтения следующего символа, необходимо приостановить операцию чтения в МП до окончания продвижения ленты. Это означает, что следует перевести систему в состояние *M3·T3* до момента появления следующего символа. Как будет показано в гл. 3, наиболее эффективным способом синхронизации в этом случае является выключение синхросигнала вплоть до момента ответа внешнего устройства на запрос. Однако для современных МП, за исключением небольшого числа (*Signeties 2650* и *RCA* 1800), отключить синхронизацию нельзя вследствие потери внутреннего состояния. Для преодоления этого недостатка в МП предусмотрена система синхронизации, позволяющая переводить систему в состояние ожидания на любое время без отключения синхронизации.

Для продвижения перфоленты, рассмотренной выше, можно предложить следующий способ синхронизации МП и считывателя.

Из состояния *M3·T2* (рис. 2.4) МП переводится в состояние ожидания $M3 \cdot T_w$, как показано на рис. 2.5, *а*. В этом состоянии посылается запрос на внешнее устройство, в данном случае считыватель. Когда внешнее устройство выдает сигнал ответа, сигнал на его выводе *«Готовность»* становится нулевым и остается

таким, пока внешнее устройство выдает ответ. Этот сигнал снова становится единичным после того, как внешнее устройство выдаст ответ. Переход сигнала r из 0 в 1 используется для переключения МП из состояния ожидания в состояние $M3 \cdot T3$

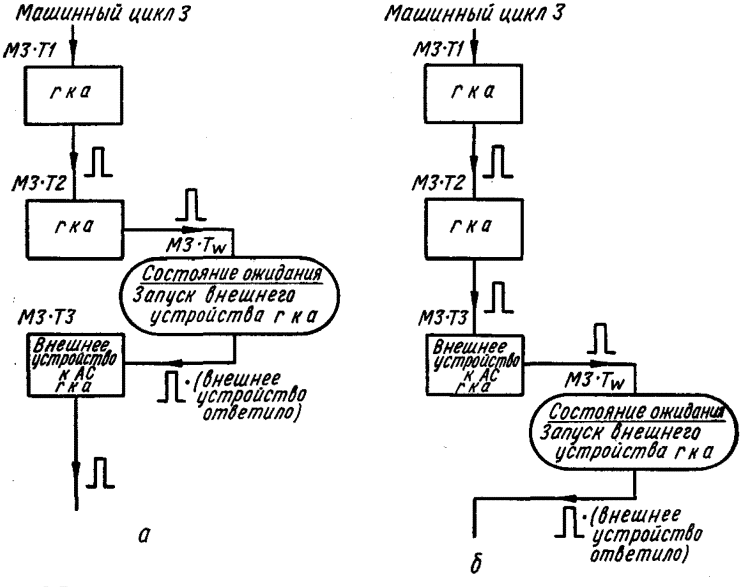


Рис. 2.5

(рис. 2.5, *а*). Этот переход позволяет в дальнейшем продолжить нормальное функционирование схемы (чтение ленты и т. д.).

Если для чтения перфоленты вначале выполнить считывание, а затем продвижение ленты, то состояние ожидания необ-

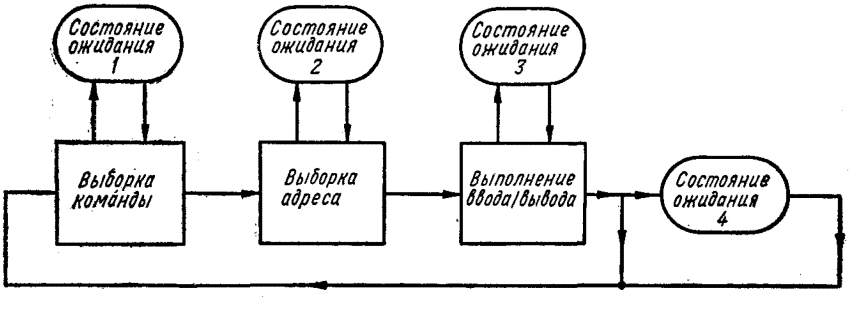


Рис. 2.6

ходимо ввести после считывания, т. е. после выхода из состояния $M3 \cdot T3$ (рис. 2.5, *б*).

Когда МП входит в состояние ожидания, на внешнее устройство посылается сигнал запроса. МП находится в состоянии ожи-

дания до тех пор, пока не подготовится ответ (изменение сигнала *r* от 0 к 1). После этого МП из состояния ожидания переходит в состояние *M1·T1* (рис. 2.4), позволяющее возобновить его нормальное функционирование.

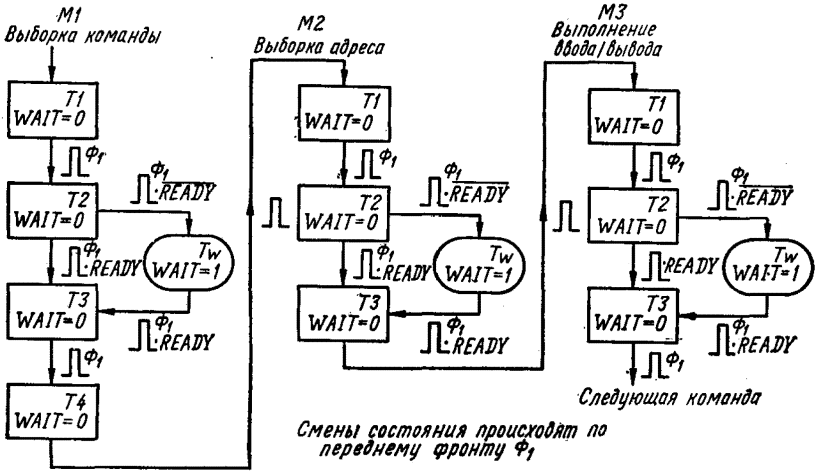


Рис. 2.7

Как видно, возможны два способа синхронизации ввода/вывода. При первом вначале на внешнее устройство подается сигнал активации (запуска), а затем производится считывание данных (в нашем примере — протяжка и чтение); во втором —

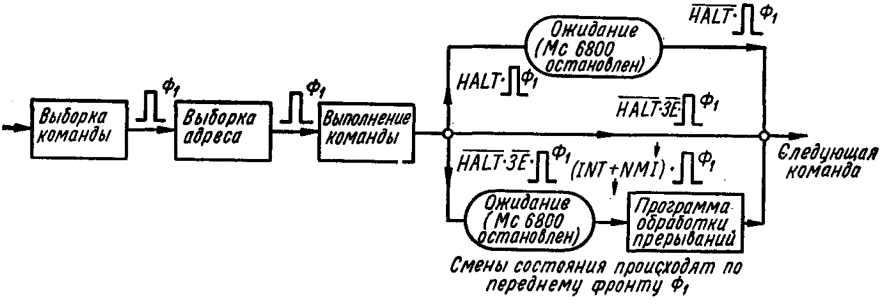
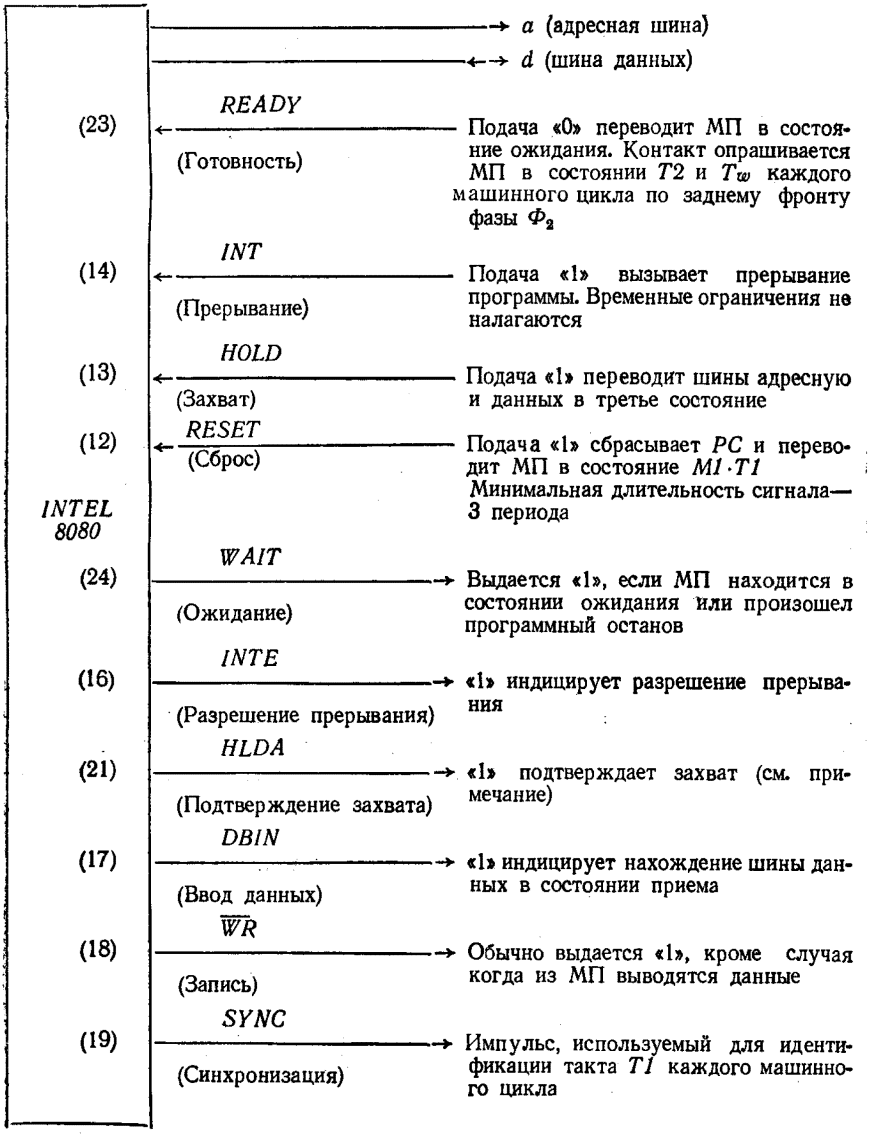


Рис. 2.8

вначале МП производит чтение, а затем выдает сигнал активации (чтение и протяжка). Будем называть эти способы соответственно: активация и доступ; доступ и активация.

Диаграмма выполнения команды ввода/вывода МП с возможностью ввода состояния ожидания в любом машинном цикле или в конце команды ввода/вывода показана на рис. 2.6. Таким



Примечание. Эти сигналы становятся равными «1» с некоторой задержкой по отношению к переднему фронту Φ_1. Адресная шина и шина данных переходят в третье состояние через некоторое время после появления фронта следующего импульса Φ_2.

Рис. 2.9

Продолжение рис. 2.9

Разряд шины данных	Информация о состоянии	Выборка команды	Чтение памяти	Запись в память	Чтение стека	Запись в стек	Чтение при вводе	Запись при выводе	Подтверждение прерывания	Подтверждение останова	Подтверждение прерывания при останове
D_0	*INTA* (Подтверждение прерывания)	0	0	0	0	0	0	0	1	0	1
D_1	$\overline{WD}$ (Запись — вывод)	1	1	0	1	0	1	0	1	1	1
D_2	*STACK* (Стек)	0	0	0	1	1	0	0	0	0	0
D_3	*HLTA* (Подтверждение останова)	0	0	0	0	0	0	0	0	1	1
D_4	*OUT* (Вывод)	0	0	0	0	0	0	1	0	0	0
D_5	*M1* (Цикл *M1*)	1	0	0	0	0	0	0	1	0	1
D_6	*INP* (Ввод)	0	0	0	0	0	1	0	0	0	0
D_7	*MEMR* (Чтение памяти)	1	1	0	1	0	0	0	0	1	0

Примечание. Слово состояния в МП *In 8080* выдается кратковременно в течение такта *T1* каждого машинного цикла по шине данных, поэтому необходимо позаботиться о его сохранении (прим. перев.).

Рис. 2.9

образом, можно осуществить синхронизацию с памятью и оба типа синхронизации с внешними устройствами.

Не все МП имеют состояние ожидания, подобное изображенному на рис. 2.6. Например, *In 8080* не имеет состояния ожидания 4, в то время как *MC 6800* имеет только состояние ожидания 4[2, 3] (рис. 2.7).

Вход в состояние ожидания и выход из него обычно осуществляются упомянутыми ранее сигналами на определенных контактах микросхемы МП. Например, для МП *In 8080* появление логического «0» на контакте *23* («Готовность») вызывает его переход в состояние ожидания, а появление логической «1» на том же контакте — выход из состояния ожидания (рис. 2.7 и 2.9). Для *MC 6800* переход в состояние ожидания вызывается подачей логического «0» на контакт *2* (линия «Останов») или подачей команды «Ожидание прерывания» (код операции 3E). Выход из состояния ожидания осуществляется подачей логической «1» на контакт *2* или генерированием запроса прерывания (рис. 2.8 и 2.10).

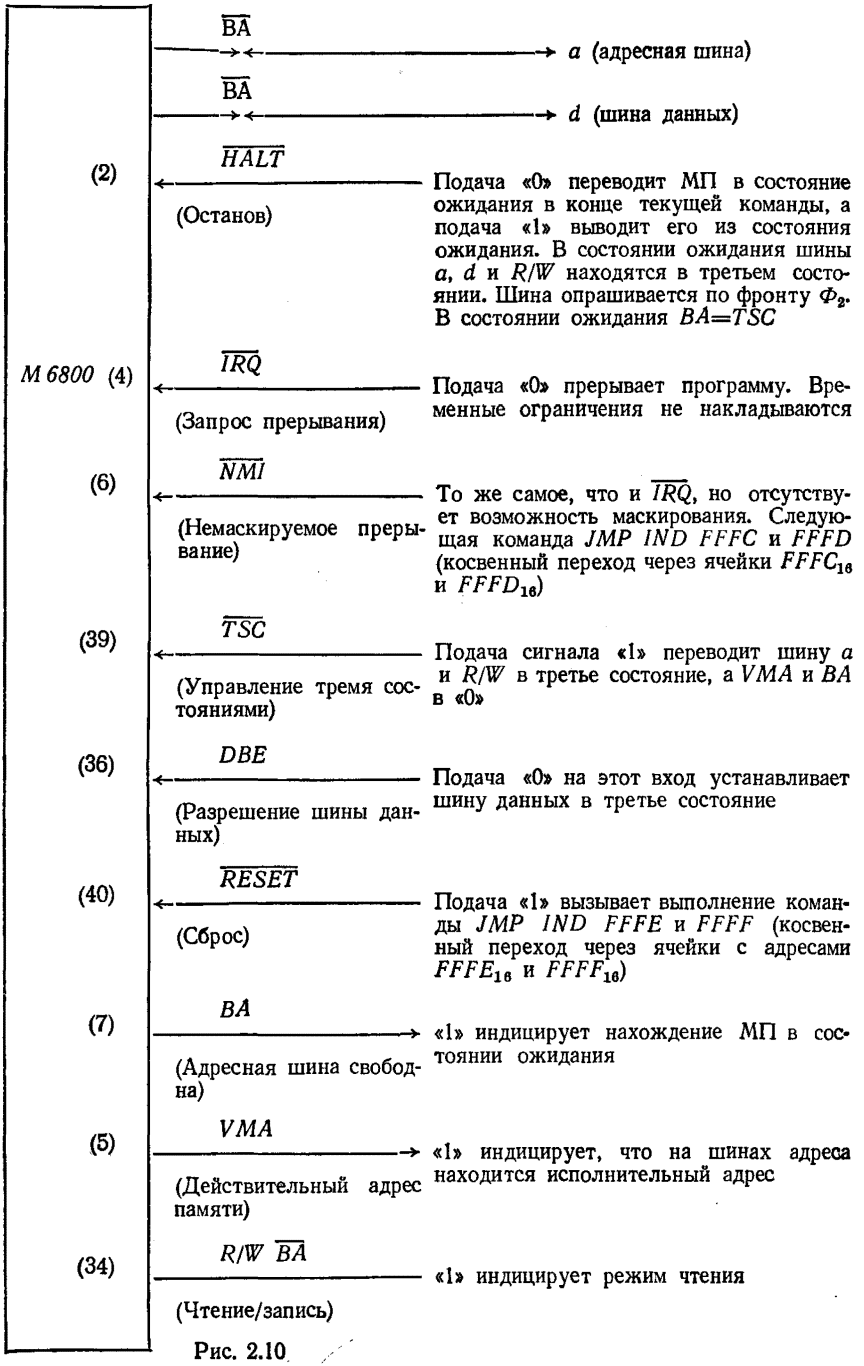

$\overline{BA}$
a (адресная шина)
$\overline{BA}$
d (шина данных)
(2)
$\overline{HALT}$
(Останов)
Подача «0» переводит МП в состояние ожидания в конце текущей команды, а подача «1» выводит его из состояния ожидания. В состоянии ожидания шины a, d и R/W находятся в третьем состоянии. Шина опрашивается по фронту Φ_2. В состоянии ожидания BA=TSC
M 6800 (4)
$\overline{IRQ}$
(Запрос прерывания)
Подача «0» прерывает программу. Временные ограничения не накладываются
(6)
$\overline{NMI}$
(Немаскируемое прерывание)
То же самое, что и $\overline{IRQ}$, но отсутствует возможность маскирования. Следующая команда JMP IND FFFC и FFFD (косвенный переход через ячейки $FFFC_{16}$ и $FFFD_{16}$)
(39)
$\overline{TSC}$
(Управление тремя состояниями)
Подача сигнала «1» переводит шину a и R/W в третье состояние, а VMA и BA в «0»
(36)
DBE
(Разрешение шины данных)
Подача «0» на этот вход устанавливает шину данных в третье состояние
(40)
$\overline{RESET}$
(Сброс)
Подача «1» вызывает выполнение команды JMP IND FFFE и FFFF (косвенный переход через ячейки с адресами $FFFE_{16}$ и $FFFF_{16}$)
(7)
BA
(Адресная шина свободна)
«1» индицирует нахождение МП в состоянии ожидания
(5)
VMA
(Действительный адрес памяти)
«1» индицирует, что на шинах адреса находится исполнительный адрес
(34)
R/W $\overline{BA}$
(Чтение/запись)
«1» индицирует режим чтения

Рис. 2.10

2.3. СИГНАЛЫ МИКРОПРОЦЕССОРА

Как и для всех последовательностных схем, работа микропроцессора характеризуется внутренними состояниями и соответствующими внешними сигналами. Эти внешние сигналы будем называть сигналами управления. Для обеспечения синхронизации внешних устройств с МП последний формирует сигналы состояний. Эти сигналы не отражают непосредственно внутреннего состояния МП, как это было бы при использовании переменных состояния (вторичных сигналов).

Для того чтобы правильно использовать сигналы состояния МП, разработчик должен глубоко понимать их смысл. Это особенно важно для МП. Объясняется это тем, что шины адреса и данных в МП используются в течение командного цикла по принципу разделения времени несколькими компонентами МП, в связи с чем важны временные соотношения между сигналами управления и сигналами состояния.

Сигналы управления и сигналы состояния вместе будем обозначать как микропроцессорные сигналы. Эти сигналы у различных МП существенно отличаются. Примером тому может служить сравнение МП *Intel 8080* и *Motorola 6800*.

2.4. РЕЖИМЫ РАБОТЫ МИКРОПРОЦЕССОРА

Системы МП могут работать в одном из следующих режимов: 1) внутреннем, 2) ожидания/счета, 3) проверки и пропуска, 4) прерывания, 5) прямого доступа к памяти (ПДП), 6) прямой передачи данных (ППД). Рассмотрим подробно все режимы.

Внутренний режим. В этом режиме команды и данные расположены в системных ПЗУ и ОЗУ. Так как в этом режиме нет обращения к внешним устройствам, он не представляет для нас интереса и в дальнейшем рассматриваться не будет. Заинтересованный читатель может найти подробное описание в работе [1].

Режим ожидания/счета *. В этом режиме внутренние операции МП синхронизируются с более медленными ответами от внешних устройств путем перевода МП в состояние ожидания до тех пор, пока выбираемое устройство не ответило на запрос (см. рис. 3.3). Перевод МП в состояние ожидания был изложен выше. Режим ожидания/счета характеризуется следующим:

1) отсутствием проблемы временной синхронизации в обычной форме;

2) время, затрачиваемое на разработку, минимально по сравнению с другими режимами;

3) аппаратурный интерфейс минимален (два проводника — «запрос» и «ответ»);

* Этот режим используется при организации непрограммного асинхронного обмена (прим. ред. пер.).

4) концепции режима ожидания/счета легкодоступны для понимания;

5) сопровождение таких систем несложно.

Недостатком этого режима по сравнению с режимом проверки и пропуска является то, что МП простаивает в ожидании ответа выбранного устройства. В некоторых случаях это не только нежелательно, но и недопустимо.

Режим проверки и пропуска *. Функционально этот режим похож на режим ожидания/счета со следующими отличиями. Синхронизация МП с внешним устройством (ВУ) выполняется с помощью программного цикла опроса, в течение которого ожидается ответ выбранного устройства. В этом цикле МП считывает и анализирует состояние ВУ. Если устройство не готово, то цикл опроса повторяется, в противном случае происходит выход из цикла и переход к дальнейшей отработке программы (см. рис. 4.1).

Режим прерывания. Используется для прерывания процесса выполнения программы МП путем подачи внешнего сигнала (сигнала прерывания) и выполнения некоторой последовательности команд (программы обработки прерывания), запрошенной внешним устройством. В конце программы обработки прерывания управление передается прерванной программе в точку, где произошло прерывание (см. рис. 5.1).

Как будет показано в гл. 5, проектирование и реализация этого режима, хотя и осуществляются по четкой процедуре, но требуют более сложной аппаратуры и программного обеспечения, чем для любого другого режима. Основное преимущество этого режима — высокое быстродействие.

Режим ПДП. В режиме прямого доступа к памяти (ПДП) устанавливается прямая связь между внешним устройством и памятью системы (см. рис. 6.1, *б*) с целью двунаправленной передачи данных между ними. Этот режим полезен для передачи больших блоков данных из памяти во внешнее устройство и обратно. Хотя такая передача инициируется программой, протекает она без ее участия. Обычно, хотя и не всегда, по окончании передачи данных интерфейс выставляет флажок окончания передачи блока в, чтобы сообщить программе, что указанный блок данных уже передан.

Вопреки кажущейся сложности разработка и реализация интерфейса ПДП весьма проста, в чем мы убедимся в гл. 6. Аппаратурная часть интерфейса не содержит сложностей, а программная часть для запуска передачи содержит примерно десять команд на пересылаемый блок.

Внимание читателя обращается также на то, что ПДП применяется для обмена с быстродействующими устройствами и не применяется для таких медленных устройств, как перфоленточ-

* Этот режим используется при организации программного асинхронного обмена (прим. ред. пер.).

ный считыватель, перфокарточный считыватель, перфоратор и т. д. В гл. 6 будет доказано, что режим ПДП можно использовать для обмена блоками между памятью и внешними устройствами, независимо от быстродействия последних.

Режим ППД. Режим прямой передачи данных (ППД) аналогичен режиму ПДП с той лишь разницей, что связь для обмена данными устанавливается между двумя или более внешними устройствами микропроцессорной системы, которая выдает разрешение на такой обмен (см. рис. 7.3). Этот режим можно использовать, например, когда необходимо скопировать содержимое некоторого блока памяти.

Так как каждая пересылка байта данных через микропроцессор требует выполнения некоторой последовательности команд, то при пересылке под управлением МП будет потеряно время, даже если при этом обработка данных в МП не производится. Поэтому в подобных случаях применяют прямую передачу данных.

Формальный подход к реализации этого режима основывается на использовании последовательностных уравнений, которые обсуждались в гл. 1.

2.5. ПОЛУПРОВОДНИКОВАЯ ПАМЯТЬ

Полупроводниковая память обычно выпускается в виде интегральных схем в корпусе типа DIP. Каждый такой корпус содержит определенное число элементов для хранения битов информации: число этих элементов обычно равно степени двух. Биты могут быть организованы в группы (байты — последовательности бит, трактуемые как единое целое), длина которых кратна восьми (2^3) или меньше. Микросхема памяти, организованная как $2^m \cdot 2^n$ матрица, содержит 2^{m+n} ячеек для хранения 2^m байт длиной по 2^n бит. Такая микросхема имеет m адресных линий, позволяющих адресовать каждый байт в отдельности. Например, микросхема памяти объемом 128×8 ($2^7 \cdot 2^3$) содержит 1024 (2^{10}) бита, организованных как 128 8-битовых байтов. Для такой микросхемы необходимо 7 адресных линий для адресации произвольного байта. Аналогично организация 4096×1 описывает микросхему памяти емкостью 4096 бит с 12 адресными линиями.

Кроме адресных выводов, микросхема памяти обычно имеет выводы выборки микросхем, которые используются для идентификации микросхемы при модульной организации памяти.

Способ распределения битов адресного кода, выдаваемого МП, между выводом выборки микросхемы и адресными выводами определяется архитектурой системы.

Полупроводниковая память подразделяется на несколько типов: ПЗУ, СППЗУ, ППЗУ и ОЗУ. Краткое их описание приведено ниже.

Постоянное запоминающее устройство (ПЗУ) — это память, содержимое в которую записывается при изготовлении. Это содержимое хранится там постоянно и не может быть стерто или изменено. Благодаря этому ПЗУ, как правило, служит для хранения стандартных программ, таблиц и т. д. В отличие от ОЗУ, ПЗУ сохраняет содержимое при выключении питания.

Основные преимущества ПЗУ — большая емкость, малая потребляемая мощность, малое время выборки и сохранность информации при выключении питания, недостаток — неизменяе-

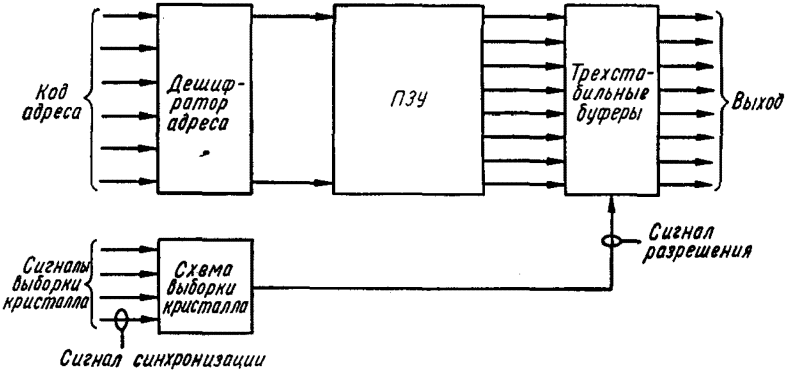


Рис. 2.11

мость записанной в них информации, поэтому одна ошибка обходится очень дорого при серийном производстве.

Структурная схема ПЗУ показана на рис. 2.11. Выдача выбранного байта на шину данных МП через трехстабильные буферы синхронизируется сигналом разрешения выдачи. Естественно, что этот сигнал должен подаваться с учетом времени выборки микросхемы памяти. Чаще всего для этой цели используется один из входов выборки микросхемы.

Стираемое и программируемое постоянное запоминающее устройство (СППЗУ). После установки в системе такое устройство ведет себя так же, как и ПЗУ. Различие между ними заключается в том, что СППЗУ может программировать сам пользователь. Это означает, что его содержимое можно стереть и заменить новой информацией. Процесс перепрограммирования требует специальной аппаратуры, такой, например, как источник ультрафиолетового излучения (для стирания) и источник высоковольтных импульсов (для перезаписи). После того как первоначальная информация стерта, новая информация записывается путем подключения каждой его ячейки к высоковольтному источнику. Благодаря этому происходит заряд соответствующих емкостей, обладающих возможностью сохранять заряженное состояние около 100 лет.

СППЗУ на практике используется для запоминания редкоизменяемой или неизменяемой информации, заносимой самим пользователем.

Программируемое постоянное запоминающее устройство (ППЗУ) в системе ведет себя так же, как и ПЗУ. Для записи «1» в выбранную ячейку памяти через нее пропускается ток, достаточный для пережигания плавкой перемычки. Таким образом, создается постоянно разомкнутая цепь, причем этот процесс может быть проведен только один раз. ППЗУ не так на-

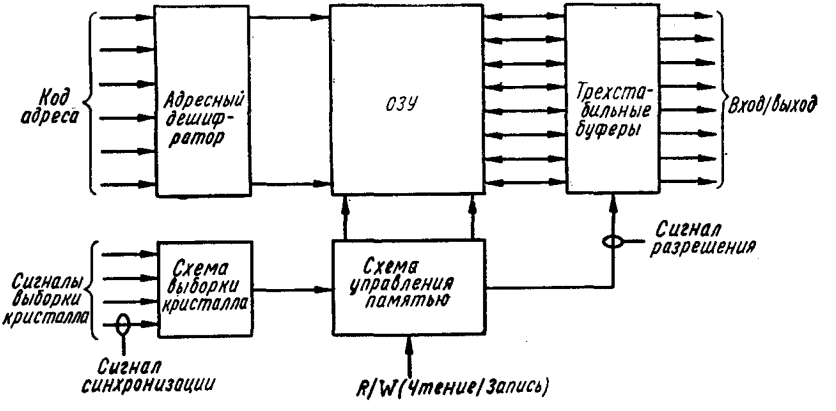


Рис. 2.12

дежны, как СППЗУ, в особенности для управления процессами и промышленного применения.

Оперативное запоминающее устройство (ОЗУ). Содержимое может считываться и записываться под управлением программы. Структурная схема ОЗУ изображена на рис. 2.12. Схема имеет адресные входы для выборки определенной ячейки ОЗУ и входы выборки микросхемы в системе памяти.

Для управления ОЗУ обычно используются два сигнала: сигнал чтение/запись, определяющий выполняемое действие, и сигнал разрешения для управления временем выдачи/приема аналогично ПЗУ.

В различных ОЗУ управляющие сигналы различны.

Информация, хранящаяся в ОЗУ, пропадает при выключении питания. Некоторые ОЗУ, однако, могут работать с пониженным потреблением мощности, причем переход к такому потреблению определяется специальным управляющим сигналом. В этом режиме доступ к ОЗУ запрещен, но информация сохраняется при уменьшенной мощности.

Различают статические и динамические ОЗУ. В статическом ОЗУ записанная информация при наличии питания сохраняется без каких-либо дополнительных операций. В динамическом ОЗУ записанная информация исчезает, если через несколько милли-

секунд она не будет регенерирована. Это связано с утечкой зарядов в емкостях, служащих для хранения информации. Исчезающий заряд должен быть восстановлен до того, как произойдет утечка заряда.

Статические ОЗУ имеют меньшие емкость и быстродействие, чем динамические, но они не требуют схем регенерации. Динамические ОЗУ, вообще говоря, менее надежны, чем статические.

Стек — блок соседних ячеек ОЗУ, доступный с одного конца по принципу «вошедший первым выходит последним». В МП системах адрес стека содержится в указателе стека. Последним является реверсивный счетчик, содержимое которого обычно уменьшается на 1 перед каждой операцией записи (занесения в стек) и увеличивается на 1 после каждой операции извлечения из стека.

В каждой конкретной системе под стек отводится блок последовательных ячеек. Если начальный адрес блока не заносится автоматически в указатель стека, то это должен предусмотреть пользователь при составлении программы.

2.6. ПОРТЫ ВВОДА/ВЫВОДА

В структурной схеме на рис. 2.13, *а* показано подключение входного и выходного портов к двунаправленной шине данных. Если устройство выбрано адресной шиной и подан сигнал $e_1=1$, оно подключается к шине данных для выдачи информации. Ана-

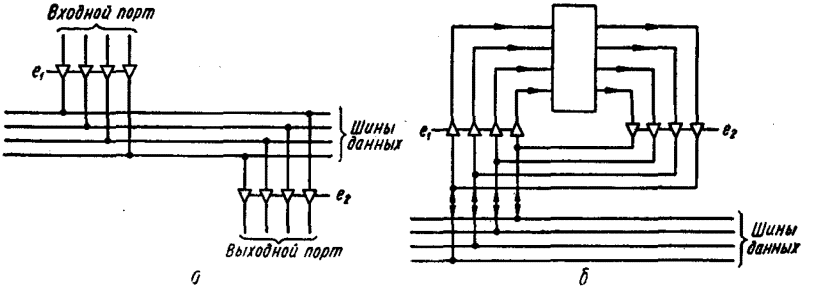


Рис. 2.13

логично, если $e_2=1$, то выбранное устройство подсоединяется через порт ввода к шине данных для приема информации (когда сигнал разрешения e равен нулю, трехстабильные буферы ведут себя, как разомкнутые цепи). Очевидно, что пока информация пересылается от одного устройства к другому, все другие устройства, подключенные к шине данных, должны быть отключены.

Схема трехстабильного буфера, подключаемого к шине данных устройства и способного работать как на ввод, так и на вывод, показана на рис. 2.13, *б*. Если $e_1=1$, а $e_2=0$, устройство

принимает данные с шины, а если $e_1=0$ и $e_2=1$,— выдает данные на шину; условие $e_1=e_2=0$ — условие отключения от шины. Применение портов ввода/вывода в МП системах показано на рис. 2.1.

В рассматриваемой системе используется только одна шина как при чтении, так и при записи. Такие шины называют двунаправленными.

2.7. АДРЕСНЫЕ ДЕШИФРАТОРЫ

В схеме на рис. 2.1 для простоты использованы отдельные адресные дешифраторы. Они представляют собой элементы *И*, выдающие выходной сигнал, когда на их вход подается определенная комбинация сигналов. Например, дешифратор адреса 6

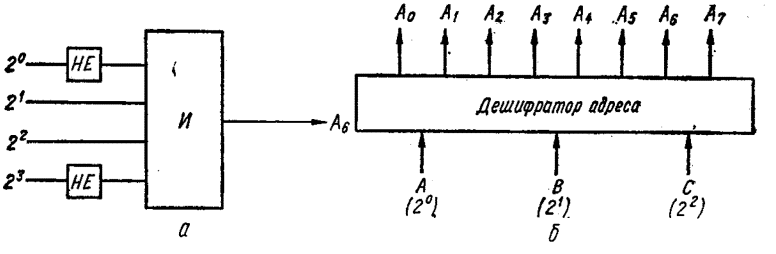


Рис. 2.14

на 4-битовом адресном поле состоит из двух инверторов и элемента *И* (рис. 2.14, *а*).

На практике для этих целей используются интегральные схемы дешифраторов. Типичный дешифратор имеет три входных линии и восемь выходных, т. е. декодирует 8 различных адресов (рис. 2.14, *б*). Большее число адресных линий можно декодировать с помощью соответствующим образом соединенных микросхем дешифратора, применяя методы, описанные в гл. 1.

2.8. ИНТЕРФЕЙСЫ

Интерфейсы предназначены для анализа состояния двух или более устройств, между которыми пересылаются данные, и для передачи последовательности сигналов управления для этих устройств. Последовательность передачи сигналов в некоторых системах программируется. Структурная схема интерфейса между источником и приемником показана на рис. 2.15.

Формально интерфейс определяется как набор цепей, сигналов и алгоритмов, необходимых для передачи данных между цифровыми устройствами.

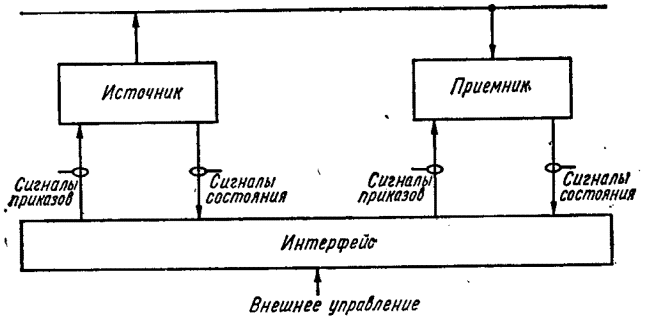


Рис. 2.15

2.9. МЕТОДИКА ПРОЕКТИРОВАНИЯ

Предлагаемая методика проектирования позволяет неквалифицированному разработчику создавать надежные и простые системы и в то же время совершенствовать свои знания при работе с более сложными системами. Изящность решения не является целью разработки, но может быть получена без особых усилий.

При проектировании МП систем необходимо учитывать следующие требования:

1. *Надежность системы.* Все системы должны функционировать правильно.
2. *Простота обслуживания.* Системы должны быть просты в обслуживании.
3. *Затраты на проектирование.* Должны быть минимальны, чтобы облегчить творческий подход к проекту.
4. *Документация.* Должна быть краткой и точной. Диаграммы и символы должны быть строго определены, они должны легко пониматься зарубежными потребителями.
5. *Процедура проектирования.* Должна быть простой для реализации технической идеи. В нашем случае никаких дополнительных теоретических знаний не требуется.
6. *Модификации.* Системы должны легко преобразовываться для новых условий.

2.10. ПРОЦЕДУРА ПРОЕКТИРОВАНИЯ [1]

Процедура проектирования состоит из пяти шагов (рис. 2.16).

Шаг 1 — формирование технического задания. Спецификация проектируемой системы выражается разработчиком в принятых терминах, чтобы гарантировать полноту учета всех требований к системе.

Этот шаг очень важен для успешного взаимодействия между разработчиком системы и потребителем. Упущения на этом шаге

обычно служат причиной неработоспособности системы и приводят впоследствии к необходимости последующего перепроектирования.

Шаг 2 — внешнее проектирование. Разработчик изучает выходные характеристики устройств, которые будут использованы. При этом следует избегать любого рассмотрения внутренних характеристик.

Шаг 3 — системное проектирование. Разработчик детализирует характеристики системы в общих чертах при помощи структурных схем и временных диаграмм.

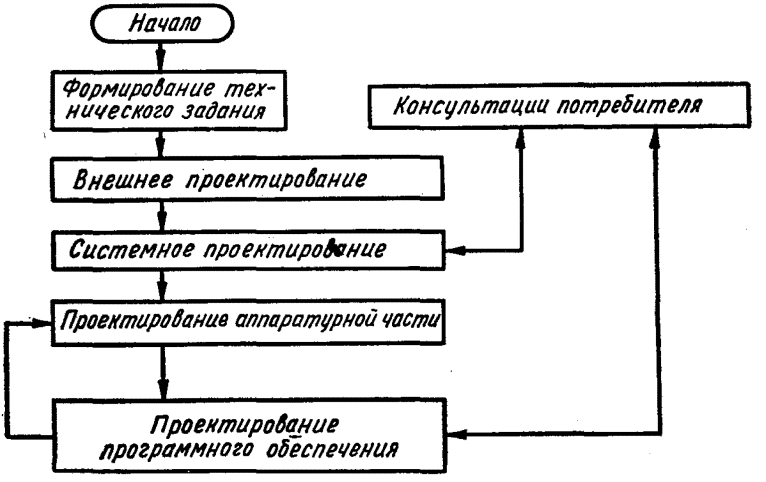


Рис. 2.16

Шаг 4 — проектирование аппаратурной части. Результат 4-го шага неокончателен и может быть изменен на 5-м шаге. 4-й и 5-й шаги многократно повторяются по известной методике последовательного приближения на основе хорошо проверенных методов [4].

Шаг 5 — проектирование программного обеспечения [5]. На основе аппаратурного решения на 4-м шаге и набора команд машины создается программное обеспечение системы. Этот процесс хорошо выявляет улучшения, которые можно ввести в аппаратурные средства на 4-м шаге. Фактически 4-й и 5-й шаги можно рассматривать как дополняющие друг друга, они должны повторяться до тех пор, пока не будет получено удовлетворительное решение.

2.11. СПИСОК ЛИТЕРАТУРЫ

1. Zissos D., Duncan F. G. Digital Interface Design (2nd edition). Oxford University Press.

2. INTEL 8080 Microprocessor User's Manual, September, 1975.

3. **M6800 Microprocessor** System Design Data. Motorola, 1976.
4. **Zissos D.** Problems and Solutions in Logic Design. Oxford University Press, 1976.
5. **Duncan F. G.** Microprocessor Programming and Software Design. Prentice-Hall, 1979.

ГЛАВА 3

СИСТЕМА ОЖИДАНИЯ/СЧЕТА

В этой главе излагается концепция режима ожидания/счета и его использование для проектирования МП систем (МПС). Для проектирования систем ожидания/счета не требуется специальных знаний в электронике или технике МП и поэтому его могут провести неспециалисты в этих областях. Методика проектирования изложена в соответствии с алгоритмом, описанным в параграфах 2.9 и 2.10.

3.1. БАЗОВЫЕ ОПРЕДЕЛЕНИЯ

Как было отмечено в гл. 2, в течение выполнения операций ввода/вывода важное значение имеет синхронизация цикла МП с внешним устройством. Например, если МП выдает 1 байт информации каждые 10 мкс, а в приемнике (печатающем устройстве) для печати каждого байта необходимо 100 мкс, неизбежна потеря 9 байтов из 10, если скорость работы МП не уменьшится. Таким образом, при проектировании МПС необходимо предусмотреть, чтобы МП не загружал внешнее устройство (ВУ) бóльшим потоком информации, чем оно может обработать.

Синхронизация МП с внешним устройством при таких условиях обычно достигается (как и в ЭВМ) введением программного цикла ожидания готовности устройства. Этот метод, рассмотренный в следующей главе, будем в дальнейшем называть методом проверки и пропуска.

В тех случаях, когда синхронизацию ввода/вывода можно выполнить снижением тактовой частоты микропроцессора, используют метод, известный как удлинение синхроимпульса *(clock stretching)* [1]. Этот метод рассматривается также в следующей главе. Наиболее прогрессивным является третий метод синхронизации ввода/вывода, разработанный Д. Зиссосом и Ф. Г. Дунканом [2]. Согласно этому методу, синхронизация внутренних операций МП с медленными внешними устройствами осуществляется автоматически, при этом отпадает необходимость во

внешних синхронизирующих сигналах. Благодаря этому МПС легко реализовать с достаточной надежностью. Кроме того, такие системы могут проектировать разработчики, не обладающие специальными знаниями в электронике или технике МП, например, физики, химики, инженеры-механики, медики и т. д.

Системы ожидания/счета должны удовлетворять следующим требованиям с точки зрения системного разработчика:

1. Минимальным временем на разработку.
2. Минимальным аппаратурным интерфейсом. Он состоит из двух проводников запроса и ответа (см. приложение 1).
3. Проблемы временной синхронизации практически отсутствуют.
4. Быстродействие такой системы сравнимо с получаемой по методу проверки и пропуска.
5. Малым объемом программного обеспечения.

С точки зрения пользователя:

1. Высокой надежностью вследствие малого объема аппаратуры.
2. Простотой концепции ожидания/счета.
3. Легкостью проектирования систем ожидания/счета без специальных знаний электроники, т. е. неквалифицированный пользователь может разработать и создать свою собственную систему.
4. Простотой и доступностью алгоритма функционирования.
5. Возможностью внедрения принципа ожидания/счета в уже существующие системы.
6. Простотой сопровождения.

3.2. КОНЦЕПЦИЯ ОЖИДАНИЯ/СЧЕТА

При отработке команды ввода/вывода микропроцессор переходит в состояние ожидания. Это состояние, как следует из гл. 2, характеризуется приостановкой выполнения внутренних функций МП без выключения синхронизации. При переходе МП в состояние ожидания сигнал w на линии ожидания изменяется от «0» до «1» (рис. 3.1).

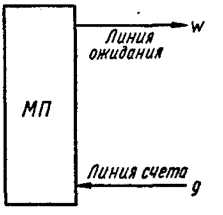


Рис. 3.1

Выход из состояния ожидания инициируется сигналом на линии счета g. В нашем случае МП выходит из состояния ожидания после изменения сигнала на линии g от «0» до «1».

3.3. СИСТЕМЫ ОЖИДАНИЯ/СЧЕТА

Структурная схема системы ожидания/счета показана на рис. 3.2. Система работает следующим образом. При выборке команды ввода/вывода микропроцессор автоматически переходит в состояние ожидания, а на внешние устройства посылается сигнал запроса. МП остается в этом состоянии до тех пор, пока

внешнее устройство не ответит на запрос, после чего он возвращается в состояние счета, замыкая цикл (рис. 3.3).

Как указывалось ранее, интерфейс типа запрос/ответ (см. приложение 1) состоит (рис. 3.2) из двух проводников [2]. Ниже приводится описание такой системы.

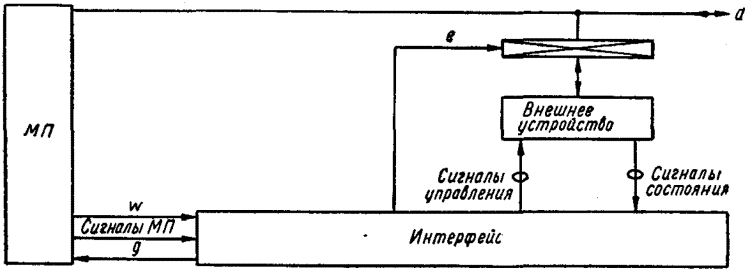


Рис. 3.2

Начнем со схемы, изображенной на рис. 3.4. Сигналы *w*, *g*, *a*, *r* имеют следующие значения:
сигнал *w:* «1» на этом выводе (линия ожидания) индицирует нахождение МП в состоянии ожидания;

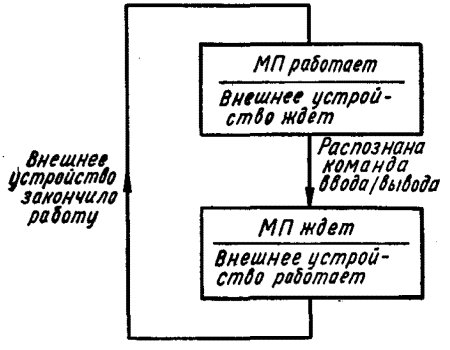


Рис. 3.3

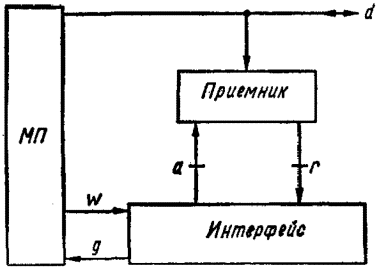


Рис. 3.4

сигнал *g:* переход из «0» в «1» на этом выводе (линия счета) выводит МП из состояния ожидания;
сигнал *a:* переход от «0» до «1» на соответствующей линии переводит устройство во включенное состояние (активирует его);
сигнал *r:* до тех пор, пока внешнее устройство готовит ответ, $r=0$; когда внешнее устройство закончит ответ, оно изменяет сигнал *r* с «0» на «1». Очевидно, что новое обращение недопустимо, пока $r=0$.

Диаграмма состояний, соответствующая работе этой системы, показана на рис. 3.5. Используя методику минимизации числа состояний (см. параграф 1.7), составим эквивалентную таблицу

состояний (рис. 3.6, *а*). В этой таблице можно слить три строки в одну, как показано на рис. 3.6, *б*.

Непосредственно из таблицы состояний можно записать следующие уравнения:

$$a = wr + w\bar{r} + (\bar{w}\bar{r}) = w; \quad (1)$$

$$g = \bar{w}r + wr + (\bar{w}\bar{r}) = r. \quad (2)$$

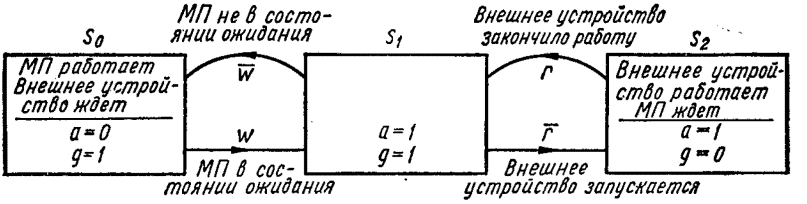


Рис. 3.5

Схема, соответствующая данной системе уравнений, показана на рис. 3.7. Как видно, интерфейс типа запрос/ответ между МП, оснащенным логикой ожидания/счета, и устройством состоит из двух проводников.

Для устройств ввода типа запрос/ответ могут потребоваться дополнительные логические схемы для формирования сигнала разрешения, управляю-

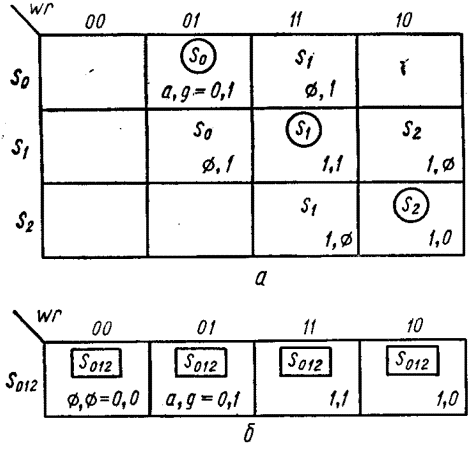


Рис. 3.6

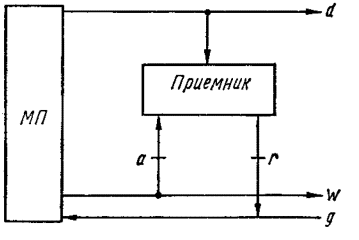


Рис. 3.7

щего портом. Синхронизация МП с устройством ввода может осуществиться введением состояния ожидания либо в течение цикла ввода/вывода, либо после него (рис. 3.8). Сигнал чтения равен «1» в течение времени выдачи внешним устройством информации в МП.

В схеме на рис. 3.8, *а* операция ввода осуществляется последовательно путем запроса устройства, а затем чтения с него. Например, для считывателя с перфоленты — это способ «протяжка на шаг и считывание». В схеме на рис. 3.8, *б* используется

обратная последовательность: сначала чтение, а затем запрос (активация). Для устройства считывания с перфоленты это — «чтение и протяжка на шаг».

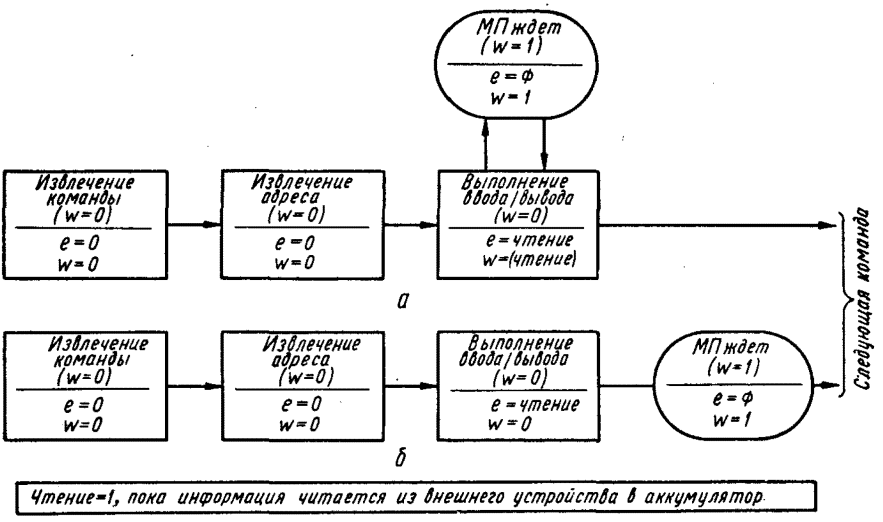


Рис. 3.8

Непосредственно из рис. 3.8, *а* следует

$$e = \text{чтение} + \text{ожидание};$$
$$w = \text{ожидание} + \text{чтение},$$

иначе

$$e = \text{ожидание} + \text{чтение} = w \qquad (3а)$$

(рис. 3.9,*а*).

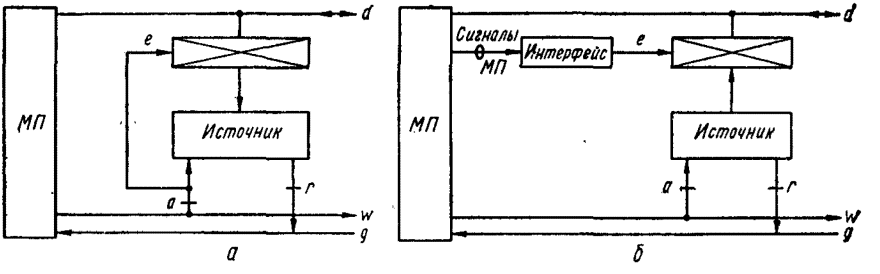


Рис. 3.9

Аналогично из рис. 3.8,*б* получим

$$e = \text{чтение} + \text{ожидание} = \text{чтение};$$
$$w = \text{ожидание}. \qquad (3б)$$

Отсюда видно, что, когда МП переходит в состояние ожидания в конце выполнения команды (рис. 3.8, *б*), двум линиям необходима дополнительная логика управления трехстабильным (ТСБ) буфером и формированием сигнала разрешения выдачи — *e* (рис. 3.9, *б*). В МП типа *MC 6800* как раз реализован этот под-

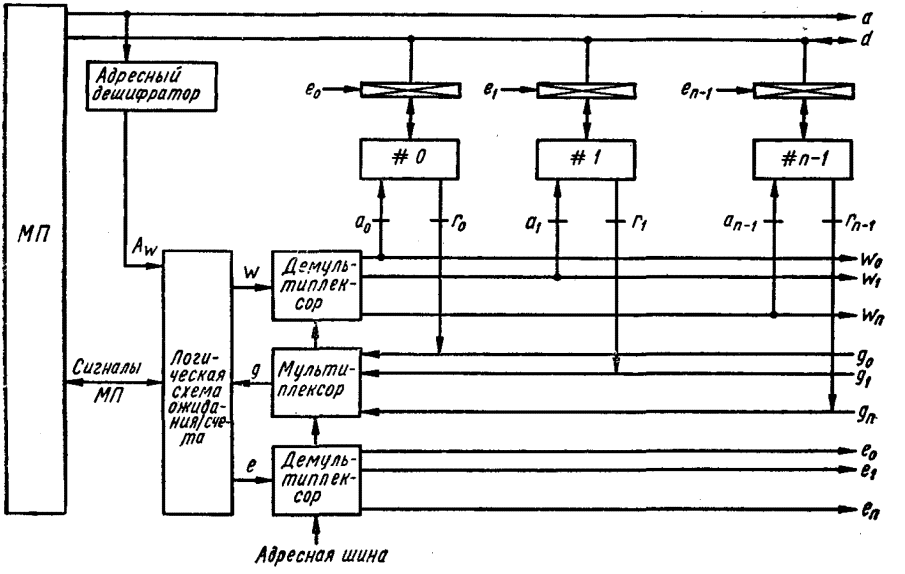


Рис. 3.10

ход, в то время как *In 8080* построен на принципе ожидания во время выполнения команды.

Для расширения системы с целью подключения n устройств преобразуем уравнения (1), (2) и (3) к следующему виду:

$$a_m = A_m w;$$
$$g = A_0 r_0 + A_1 r_1 + \ldots + A_{n-1} r_{n-1};$$
$$e_m = A_m e.$$

Схема реализации этих уравнений показана на рис. 3.10. На рис. 3.11 изображена схема упрощенной реализации системы ожидания/счета для n устройств. Очевидно, что адрес ожидаемого устройства должен постоянно выдаваться микропроцессором, пока он находится в состоянии ожидания.

Отметим, что пока информация пересылается от источника к приемнику (приемникам), все другие источники, подключенные к шине данных, должны быть отсоединены от нее, т. е. трехстабильные выходные буферы их портов должны находиться в третьем состоянии. Приемники при этом могут быть непосредственно подсоединены к шине данных своими входами.

Для некоторых многофункциональных устройств, таких, например, как накопитель на магнитной ленте, необходимо исполь-

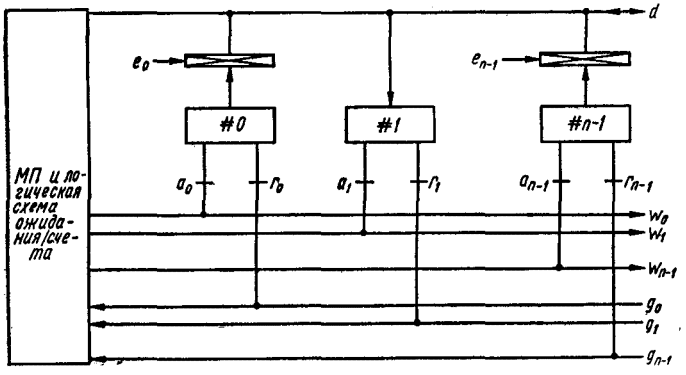


Рис. 3.11

зовать несколько пар линий ожидания/счета. Например, пара линий ожидание/счет используется для реализации обратного считывания, для разгрузки и т. д.

3.4. ЛОГИЧЕСКАЯ СХЕМА ОЖИДАНИЯ/СЧЕТА

Хотя современные микропроцессоры не предназначены для непосредственной работы в режиме ожидания/счета, они могут работать в этом режиме, если дополнить их простой логической схемой ожидания/счета, структурная схема которой изображена на рис. 3.12. Схема предназначена для поиска команды ввода/вывода, по которой осуществляется обращение к устройствам, способным вызвать возникновение ситуации ожидания (устройств с адресами A_w), и перевода микропроцессора в состояние ожидания при появлении такой команды. Пользователь, знакомый с принципами логического проектирования и имеющий практический опыт проектирования [3], легко выполняет проектирование логической схемы ожидания/счета. Основная трудность, возникающая при этом, заключается в точной интерпретации смысла сигналов микропроцессора и временных соотношений между ними.

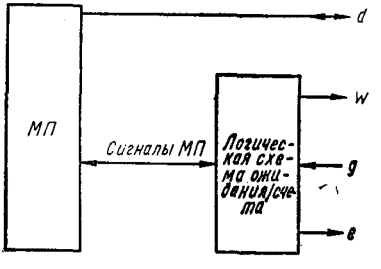


Рис. 3.12

Алгоритм проектирования рассмотрим на следующих примерах.

Пример 14. Построить логическую схему ожидания/счета для МП *In 8080*. Структурная схема микропроцессора показана на рис. 3.13,*а*, а временные ди-

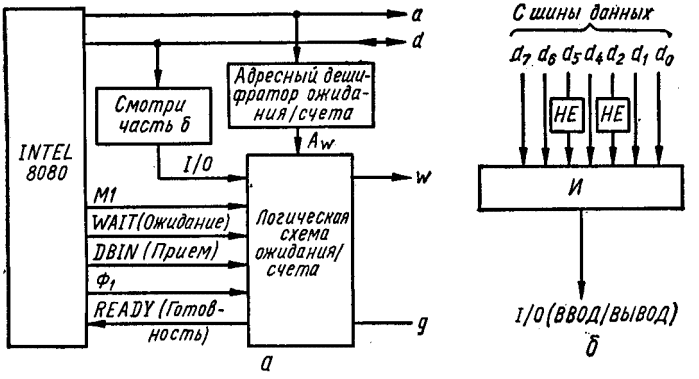


Рис. 3.13

аграммы — на рис. 3.14. Диаграмма состояния соответствующей схемы показана на рис. 3.15.

Исходное состояние схемы S_0 соответствует состоянию, когда микропроцессор функционирует, а все внешние устройства, способные работать в ре-

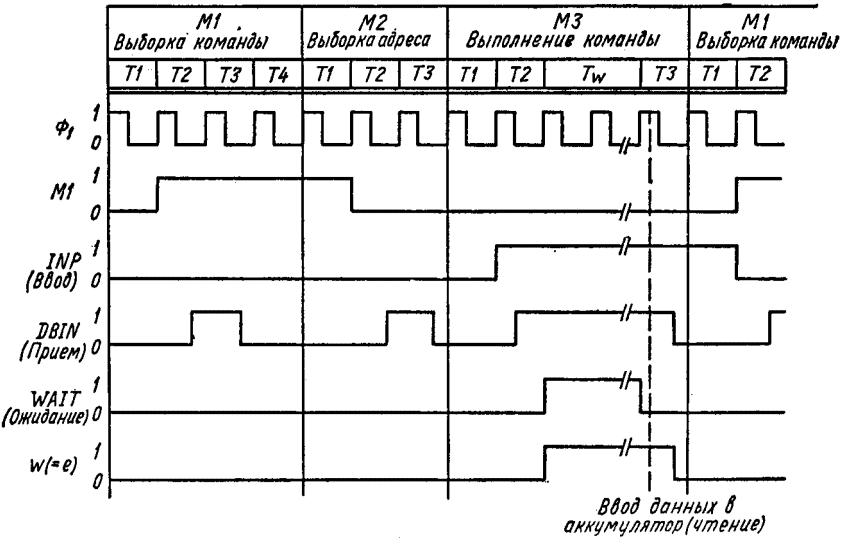


Рис. 3.14

жиме ожидания/счета, неактивны. В этом состоянии система ожидает появления команд ввода *IN* (11011011) или вывода *OUT* (11010011) при извлечении их из памяти микропроцессором. Для обнаружения этих команд применяется схема *И* (рис. 3.13,*б*), которая выдает сигнал «1» при появлении на ее входе команды ввода/вывода. Этот выходной сигнал используется для перевода системы в состояние S_1. В состояние S_2 система переходит в следующем машин-

ном цикле, когда микропроцессор находится в состоянии *M2·T3* (см. рис. 2.7).

В состоянии S_2 на линию *READY (готовность)* (рис. 3.13,*а*) подается логический «0». Благодаря этому микропроцессор через три синхропериода переходит в состояние ожидания *M3·Tw*, устанавливая при этом линию *WAIT (ожидание)* и сигнал *w* (рис. 3.13,*а*) в «1». Переход сигнала *w* в «1» активизирует внешнее устройство, адрес которого находится на шине адреса. Когда сигналы *r* и *g* на внешнем устройстве (см. рис. 3.7) находятся в состоянии логического «0», система переходит в состояние S_3. Возврат системы в состояние S_0 осуществляется по первому синхроимпульсу, пришедшему после окончания

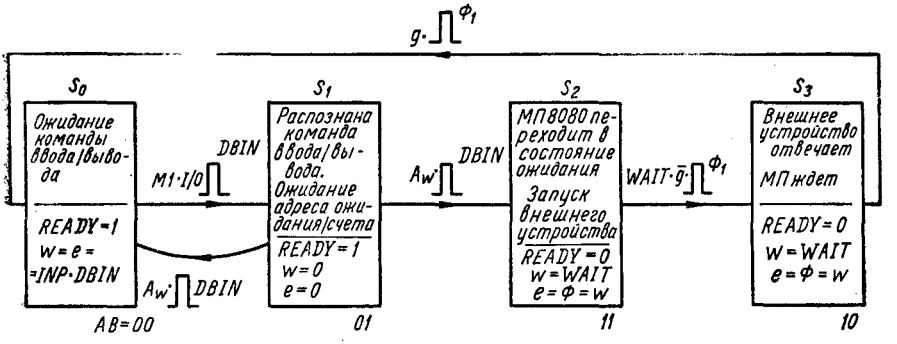


Рис. 3.15

работы внешнего устройства, т. е. после того как внешнее устройство целиком ответило на запрос, установив сигналы *r* и *g* в «1».

Реализация такой диаграммы состояний весьма проста. Алгоритм, используемый для этого, описан в параграфе 1.10. Непосредственно из диаграммы состояний (рис. 3.15) получаем

$$S_A = S_1 A_w = \overline{A}BA_w, \text{ поэтому } J_A = BA_w;$$

$$R_A = S_3 g + (S_0) = A\overline{B}g + (\overline{A}\overline{B}) = \overline{B}g, \text{ поэтому } K_A = \overline{B}g;$$

$$\begin{aligned} S_B &= S_0 M1 \cdot I/O + (S_1 A_w) + (S_2 M1) + (S_3 M1) \\ &= \overline{A}\overline{B}M1 \cdot I/O + (\overline{A}BA_w) + (AB \cdot M1) + (A\overline{B} \cdot M1) \\ &= \overline{B}M1 \cdot I/O, \text{ поэтому } J_B = M1 \cdot I/O; \end{aligned}$$

$$\begin{aligned} R_B &= S_2 \mathrm{WAIT}\overline{g} + S_1 \overline{A}_w + (S_3) \\ &= AB\mathrm{WAIT}\overline{g} + \overline{A}BA_w + (A\overline{B}) \\ &= A\mathrm{WAIT}\overline{g} + \overline{A}B\overline{A}_w, \text{ поэтому } K_B = A\mathrm{WAIT}\overline{g} + \overline{A}\overline{A}_w; \end{aligned}$$

$$C = (S_0 + S_1)\,\mathrm{DBIN} + (S_2 + S_3)\,\Phi_1 = \overline{A}\mathrm{DBIN} + A\Phi_1;$$

$$\begin{aligned} \mathrm{READY} &= S_0 + S_1 \\ &= \overline{A}\overline{B} + \overline{A}B \\ &= \overline{A}, \end{aligned}$$

где DBIN соответствует сигналу команды «Ввод данных».

Поскольку микропроцессор *In 8080* входит в состояние ожидания при выполнении команд ввода/вывода в течение цикла выполнения команды (см. параграф 3.3), то

$$e = \text{ожидание} + \text{чтение} = w.$$

Для *In 8080*

$$\text{ожидание} = (S_2 + S_3)\,\mathrm{WAIT} = A\mathrm{WAIT};$$

$$\text{чтение} = \mathrm{INP\,DBIN}.$$

Подставляя эти выражения в уравнение для сигнала разрешения e, получаем

$$e = A\,\mathrm{WAIT} + \mathrm{INP}\,\mathrm{DBIN} = w.$$

Соответствующая схема логики показана на рис. 3.16. Следует обратить внимание на тот факт, что никакие дополнительные логические схемы разре-

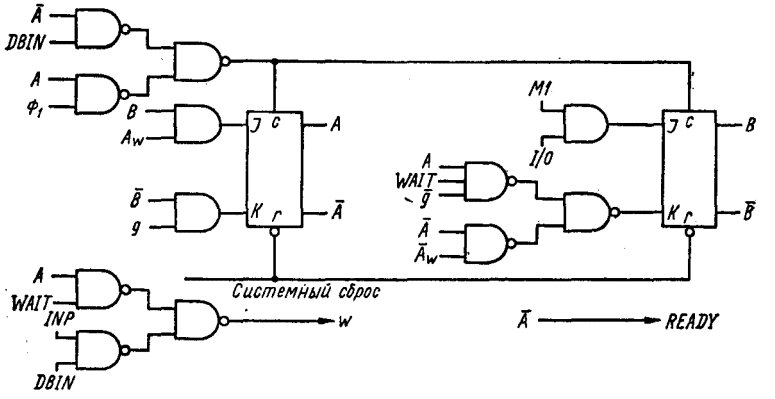


Рис. 3.16

шения в портах ввода/вывода системы ожидания/счета при использовании МП *In 8080* не требуются.

Пример 15. Спроектировать логическую схему ожидания/счета для микропроцессора *Mc6800*.

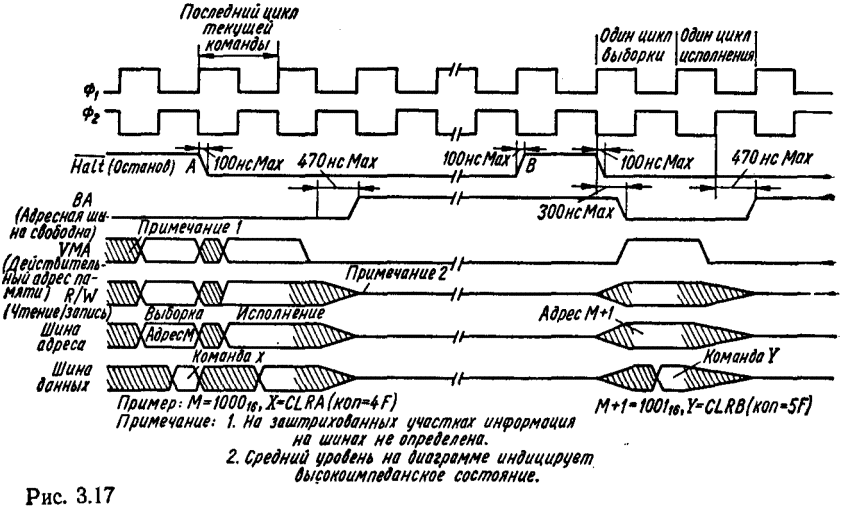


Рис. 3.17

Микропроцессор *Mc6800* останавливается в конце команды путем перевода линии $\overline{HALT}$ (контакт 2 корпуса МП) не позже, чем через 100 нс после появления фронта импульса Φ_1 в состояние «0» (рис. 3.17).

Так как выполнение команды ввода/вывода в начальных циклах не отличается от обычного обращения к памяти, то разработчики систем, как прави-

ло, располагают адреса внешних устройств в поле адресов памяти. Это позволяет определить цикл ввода/вывода по появлению адреса устройства на шине данных в течение цикла извлечения адреса или на адресной шине в течение цикла выполнения. Обращение к соответствующим временным диаграммам, приведенным в различных изданиях, к сожалению, не дает необходимого набора сигналов, позволяющего отслеживать содержимое шин данных в течение цикла извлечения адреса (автор не желает подчеркивать, что такой набор сигналов обязательно существует).

Из-за этого рекомендуется применение второго метода, т. е. анализ адресной шины в течение последнего цикла команды. Используемые для этой цели сигналы показаны на рис. 3.17 (они взяты из работы [1]). Из этой диаграммы ясно, что сигналы на адресной шине стабильны в течение последнего цикла команды, начиная от спада синхроимпульса Φ_1. Это означает, что нельзя сбросить в «0» сигнал линии $\overline{HALT}$ в течение 100 нс после появления фронта Φ_1, как это требуется в руководстве к МП. Эта проблема на практике решается путем добавления команды NOP *(нет операций)* после каждой команды ввода/вывода. Ниже приведены две возможные схемы, реализующие этот метод.

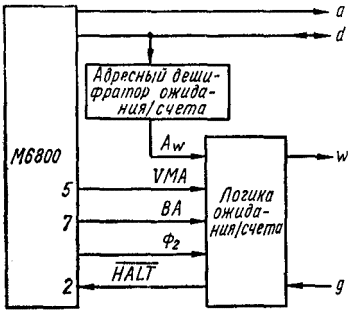


Рис. 3.18

Схема 1. Набор сигналов микропроцессора, необходимых для реализации этой схемы ожидания/счета, показан на рис. 3.18. Временные диаграммы изображены на рис. 3.17. Диаграмма состояний изображена на рис. 3.19. Функционирование схемы происходит следующим образом. Исходное состояние — S_0. Схема находится в этом состоянии, пока микропроцессор активен, а внешние устройства, работающие в режиме ожидания/счета, неактивны. Когда адрес обращения к системе ожидания/счета обнаруживается на адресных шинах в последнем цикле выполнения команды, схема переходит в состояние S_1 по синхроимпульсу Φ_2. Заметим, что, согласно рис. 3.17, сигнал на линии *VMA* и сигналы на адресной шине в течение фазы Φ_2 последнего цикла текущей команды стабильны.

В этом состоянии сбрасывается в «0» сигнал на линии МП $\overline{HALT}$. Это вызывает переход МП *Mc6800* в состояние ожидания (останова) в конце следующей команды *NOP*. Переход микропроцессора в состояние останова вызывает установку в «1» сигнала на линии *BA* (адресная шина свободна), как показано на рис. 2.8 и 2.10 (отметим, что шины данных и адреса и линия *R/W* находятся в третьем состоянии, пока МП находится в состоянии ожидания). Это вызывает установку сигнала ожидания *w* (рис. 3.7) в «1», что эквивалентно подаче запроса на внешнее устройство. Когда внешнее устройство отвечает, устанавливая

при этом сигнал готовности r, а затем и сигнал g (рис. 3.7) в «0», схема переходит в состояние S_2.

Возврат в состояние S_0 происходит при появлении первого же сигнала Φ_2, следующего за окончанием сигнала ответа внешнего устройства, о чем сообщает переход сигналов r и g в «1» (рис. 3.7).

Алгоритм построения логических схем по диаграмме состояний описан в параграфе 1.9. Ниже приведена последовательность

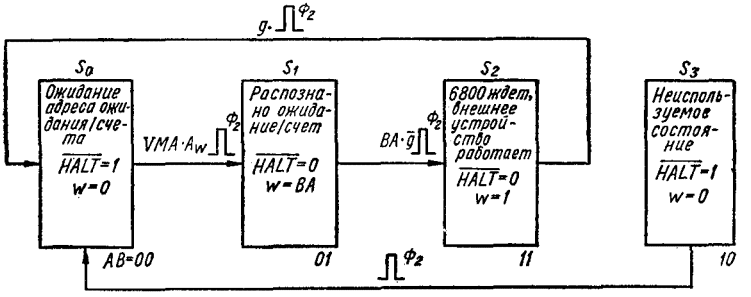


Рис. 3.19

реализации схемы по диаграмме состояний (рис. 3.19) с применением этого алгоритма:

$$S_A = S_1\bar{g}\,(BA)$$
$$= \bar{A}B\bar{g}\,(BA), \text{ поэтому } J_A = B\bar{g}\,(BA);$$
$$R_A = S_2 g + S_3$$
$$= ABg + A\bar{B}$$
$$= Ag + A\bar{B}, \text{ поэтому } K_A = g + \bar{B};$$
$$S_B = S_0\,\mathrm{VMA}\,A_w$$
$$= \bar{A}\bar{B}\,\mathrm{VMA}\,A_w, \text{ поэтому } J_B = \bar{A}\,\mathrm{VMA}\,A_w;$$
$$R_B = S_2 g$$
$$= ABg, \text{ поэтому } K_B = Ag;$$
$$\overline{\mathrm{HALT}} = S_0 + S_3 = \bar{A}\bar{B} + A\bar{B} = \bar{B};$$
$$w = S_1(BA) + S_2$$
$$= \bar{A}B(BA) + AB = B(BA) + AB.$$

Для микропроцессора *Mc6800*

$$\text{чтение} = \mathrm{VMA}\,A_w\Phi_2 R/W.$$

Подставляя эту величину в уравнение (3б), получаем

$$e = \text{чтение}$$
$$= \mathrm{VMA}\,A_w\Phi_2 R/W.$$

Соответствующая схема показана на рис. 3.20.

Схема 2. Диаграмма состояний для второго варианта логической схемы ожидания/счета *Mc6800* показана на рис. 3.21. Функционирование схемы ясно из рисунка и требуются лишь незначительные пояснения.

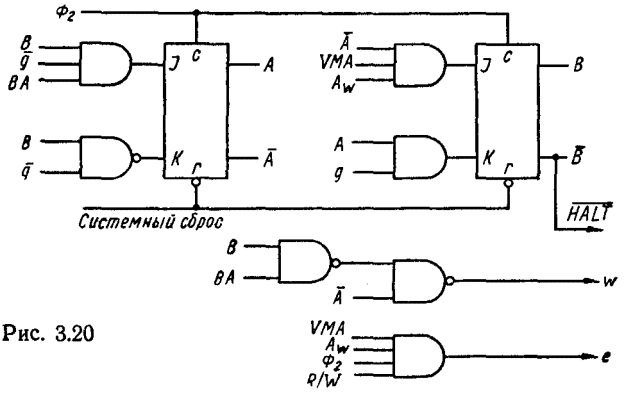


Рис. 3.20

В отличие от схемы 1, которая является синхронной, схема 2 асинхронна. Асинхронные логические схемы рассматривались в параграфе 1.9.

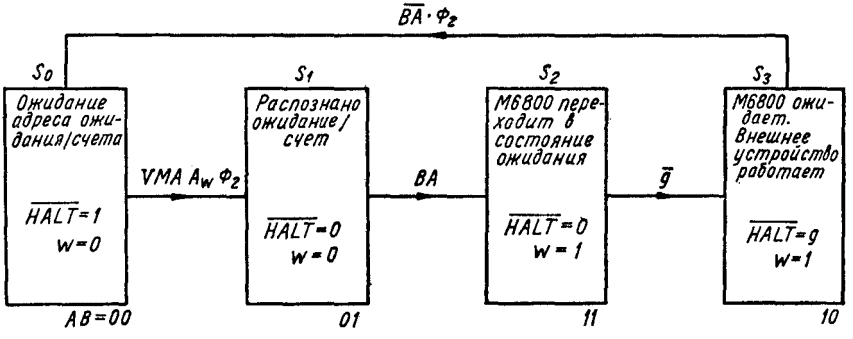


Рис. 3.21

Булевы уравнения, описывающие схему 2, получаются непосредственно из диаграммы состояний:

условие установки $A = B\,(BA)$;

условие сброса $A = \overline{B}\,(\overline{BA})\,\Phi_2 \rightarrow B + (BA) + \overline{\Phi}_2$;

условие установки $B = \overline{A}\,(\mathrm{VMA})\,A_w\Phi_2$;

условие сброса $B = A\overline{g} \rightarrow \overline{A} + g$.

Уравнения схемы 2 принимают вид

$$A = B(BA) + A(B + (BA) + \bar{\Phi}_2);$$

$$B = \bar{A}(\mathrm{VMA})\, A_w \Phi_2 + B(\bar{A} + g);$$

$$\overline{\mathrm{HALT}} = S_0 + S_3 g = \bar{A}\bar{B} + A\bar{B}g = \bar{A}\bar{B} + \bar{B}g;$$

$$w = S_2 + S_3 = AB + A\bar{B} = A.$$

Как и для схемы 1,

$$e = \text{чтение} = \mathrm{VMA}\, A_w \Phi_2 R/W.$$

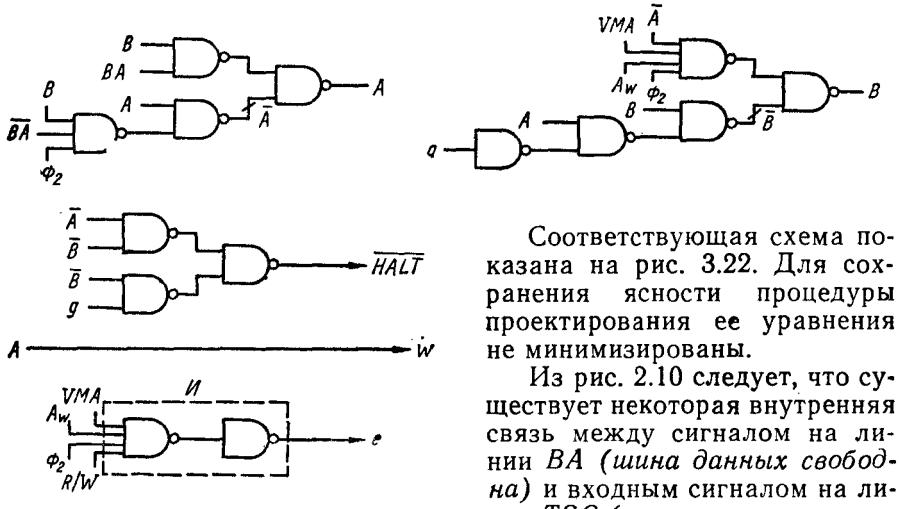


Рис. 3.22

Соответствующая схема показана на рис. 3.22. Для сохранения ясности процедуры проектирования ее уравнения не минимизированы.

Из рис. 2.10 следует, что существует некоторая внутренняя связь между сигналом на линии *BA (шина данных свободна)* и входным сигналом на линии *TSC (управление тремя состояниями)*. Установка линии *TSC* в «1» сбрасывает *BA* в «0». Это может привести к потере сигнала *w*, если сигнал на *TSC* возникнет во время отработки цикла ожидания/счета. Эту трудность можно устранить отключением контакта *39 (TSC)* на время выполнения цикла ожидания/счета. Для этого используется схема *И* с входными сигналами *TSC* и $\overline{S2+S3}$.

Тогда «контакт *39*» $= \overline{S2 + S3}\, TSC$

$= (AB + A\bar{B})\, TSC = \bar{A}\,\mathrm{TSC}.$

Состояния *S2* и *S3* описаны на рис. 3.21.

3.5. ЗАДАЧИ И РЕШЕНИЯ

В этом параграфе с помощью задач и их решений иллюстрируется методика проектирования. Обращается внимание читателя на тот факт, что, хотя примеры приведены для микропроцес-

соров *In 8080* и *Mc6800,* описанные процедуры можно применять для всех типов микропроцессоров. Особо хочется отметить, что первые три шага алгоритма проектирования осуществляются вообще безотносительно к типу микропроцессора.

Задача 1. Поиск записи. Имеется считыватель с перфоленты и МП, необходимо спроектировать систему, останавливающую

Символ	8-й код 7 бит	8 бит	Символ	8-й код 7 бит	8 бит
Пробел	040	240	:	072	272
"	042	242	;	073	273
#	043	243	=	075	275
$	044	244	?	077	277
%	045	245	*a**	100	300
&	046	246	*A*	101	301
' (кавычка)	047	247	*B*	102	302
(	050	250	*C*	103	303
)	051	251	*D*	104	304
*	052	252	*E*	105	305
+	053	253	*F*	106	306
, (запятая)	054	254	*G*	107	307
—	055	255	*H*	110	310
.	056	256	*I*	111	311
/	057	257	*J*	112	312
Перевод строки	012	212	*K*	113	313
			L	114	314
			M	115	315
Возврат каретки	015	215	*N*	116	316
Забой	177	377	*O*	117	317
0	060	260	*P*	120	320
1	061	261	*Q*	121	321
2	062	262	*R*	122	322
3	063	263	*S*	123	323
4	064	264	*T*	124	324
5	065	265	*U*	125	325
6	066	266	*V*	126	326
7	067	267	*W*	127	327
8	070	270	*X*	130	330
9	071	271	*Y*	131	331
			Z	132	332

Рис. 3.23. Коды символов

* Символ коммерческое «При».

дальнейшее считывание ленты и устанавливающую флажок при обнаружении последовательности символов 4—5—6.

Для реализации использовать режим ожидания/счета, описанный выше для микропроцессоров *In8080* и *Mc6800.*

Решение для *In 8080.*

Шаг 1 — формирование технического задания. Основная цель проектирования — построение системы просмотра принимаемых

данных для обнаружения определенных последовательностей, таких как метки, пороговые величины и т. д.

Шаг 2 — внешнее проектирование. Микропроцессор имеет логическую схему ожидания/счета, а фотосчитыватель является

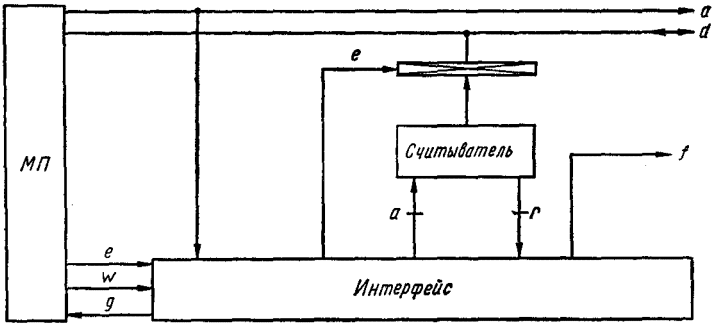


Рис. 3.24

устройством типа запрос/ответ (см. приложение 1). Информация на ленте задана в коде ASC II (рис. 3.23).

Шаг 3 — системное проектирование. Структурная схема системы показана на рис. 3.24, а алгоритм работы — на рис. 3.25.

Шаг 4 — проектирование аппаратурной части. За исключением изображенного на рис. 3.26 порта ввода/вывода никакие другие дополнительные аппаратурные средства не требуются. Это связано с тем, что для *In 8080* $e=w$, как было доказано в параграфе 3.4.

(Продолжение на стр. 83)

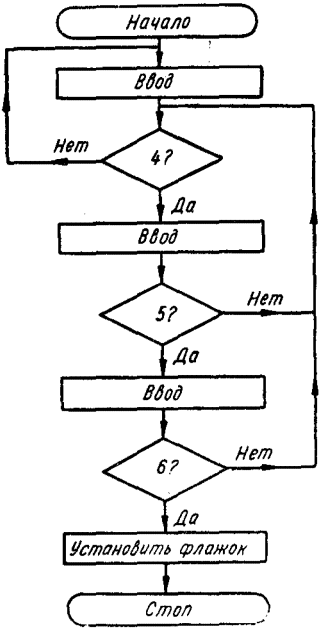


Рис. 3.25

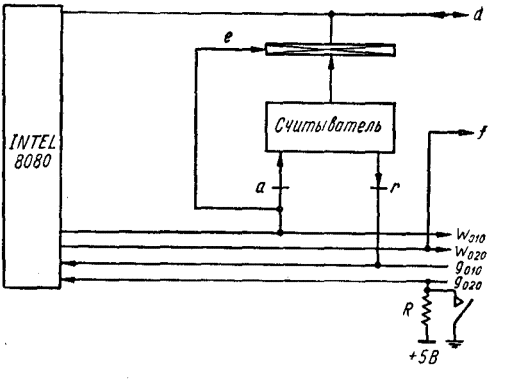


Рис. 3.26

										D6
						0				
						$D_5D_4D_3$				
		000	010	110	100	101	111	011	001
		NOP	$M_p=A$				Сдвиг		
	000	000	002	006	004	005	007	003	001
				$r:=I$	$r:=r+1$	$r:=r-1$		$P:=P+1$	$P:=II$
	010		022	026	024	025	027	023	021
			$MI:=A$				$c:=1$		
	110		062	066	064	065	067	063	061
			$MI:=HL$				BCD		
0	100		042	046	044	045	047	043	041
	$D_5D_4D_3$		$HL:=MI$				$A:=\bar{A}$		
	101		052	056	054	055	057	053	051
			$A:=MI$				$c:=\bar{C}$	$P:=P-1$	$HL:=$ $=HL+P$
	111		072	076	074	075	077	073	071
			$A:=M_p$				Сдвиг		
	011		032	036	034	035	037	033	031
	001		012	016	014	015	017	013	011
D_7	$+c$ 001	210	212	216	214	215	217	213	211
	$-c$ 011	230	232	236	234	235	237	233	231
	(—) 111	270	272	276	274	275	277	273	271
	$\forall$ 101	250	252	256	254	255	257	253	251
1	$D_5D_4D_3$			$A:=$ $=Aopr$					
	$\wedge$ 100	240	242	246	244	245	247	243	241
	$\vee$ 110	260	262	266	264	265	267	263	261
	— 010	220	222	226	224	225	227	223	221
	+ 000	200	202	206	204	205	207	203	201

Однобайтный регистр		Условие	
	r		
000	B	$\bar{z}$	не нуль
010	D	$\bar{c}$	не перенос
110	MhI	$\bar{n}$	больше или равно нулю
100	H	$\bar{p}$	нечет
101	L	p	чет
111	A	n	отрицательно
011	E	c	перенос
001	C	z	нуль

Рис. 3.27. Карта команд *INTEL* 8080

	1								
	$D_2D_1D_0$								
	001	011	111	101	100	110	010	000	
	101	103	107	105	104	106	102	100	B
		r1:=r2				*r1:=Mhl*	*r1:=r2*		
	121	123	127	125	124	126	122	120	D
						Останов			
	161	163	167	165	164	166	162	160	
	141	143	147	145	144	146	142	140	H
	151	153	157	155	154	156	152	150	L
		r1:=r2				*r1:=Mhl*	*r1:=r2*		
	171	173	177	175	174	176	172	170	A
	131	133	137	135	134	136	132	130	E
	111	113	117	115	114	116	112	110	C
	RET			*CALL*					
	311		317	315	314	316	312	310	z
		IN							
		333	337		334	336	332	330	c
	SP:=HL	*EI*	*RESTART*		Условный		Условный	Условный	
	371	373	377		374	376	372	370	n
	PC:=HL				вызов		переход	возврат	
	351	353	357		354	356	352	350	p
						A:=A op I			
	341	343	347	345	344	346	342	340	$\bar{p}$
	POP p	*DI*							
	361	363	367	365	364	366	362	360	$\bar{n}$
		OUT		*PUSH p*					
	321	323	327	325	324	326	322	320	$\bar{c}$
		JUMP							
	301	303	307	305	304	306	302	300	$\bar{z}$

Операция		Регистровая пара	Сдвиг (A)
op		*p*	
+	00	*BC*	Логический влево
—	01	*DE*	Логический вправо
∨	11	*SP**	Арифметический вправо
∧	10	*HL*	Арифметический влево
∀ *ИЛИ-НЕ*			

Сравнение (*z:=A=r*; *c:=A<r*)

—c (*A:=A—r—c*)

+c (*A:=A—r+c*)

* *AF в POP, PUSH*

Таблица 3.1

Программа поиска записи для МП *In 8080*

	8-й адрес		8-й код	16-й код	Мнемокод	Комментарии
	H	L				
L1:	003	000	333	DB	IN	Читать следующий символ
		001	010	08		
L2:		002	376	FE	CPI	Сравнить содержимое аккумулятора с числом 4
		003	264	B4		
		004	302	C2	JNZ	Если символ не равен 4, перейти к *L1*
		005	000	00		
		006	003	03	L1	
		007	333	DB	IN	Читать следующий символ
		010	010	08		
		011	376	FE	CPI	Сравнить содержимое аккумулятора с числом 5
		012	265	B5		
		013	302	C2	JNZ	Если символ не равен 5, перейти в *L2*
		014	002	02		
		015	003	03	L2	
		016	333	DB	IN	Читать следующий символ
		017	010	08		
		020	376	FE	CPI	Сравнить содержимое аккумулятора с числом 6
		021	266	B6		
		022	302	C2	JNZ	Если символ не равен 6, перейти к *L2*
		023	002	02		
		024	003	03	L2	
		025	323	D3	OUT	Поднять флажок
		026	020	10		
		027	166	76	HLT	Останов

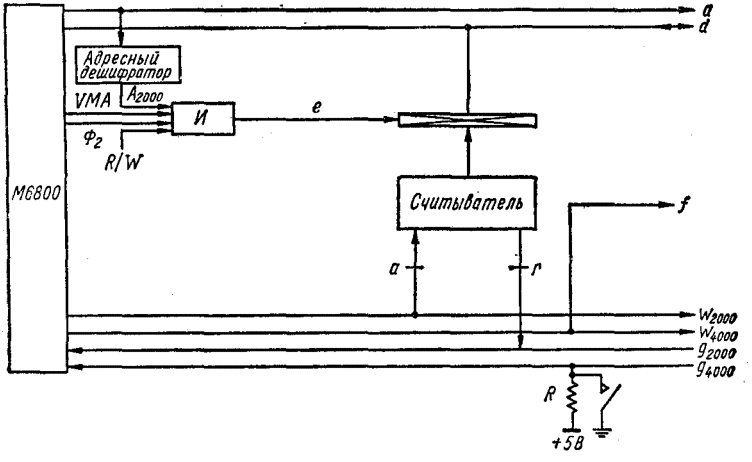


Рис. 3.29

Мнемоника	Описание	Код операции (1) D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Длительность в циклах (2)
ACI	Сложить непосредственно с *A* и переносом	1	1	0	0	1	1	1	0	7
ADCM	Сложить содержимое ячейки памяти с *A* и переносом	1	0	0	0	1	1	1	0	7
ADCr	Сложить содержимое регистра с *A* и переносом	1	0	0	0	1	*S*	*S*	*S*	4
ADDM	Сложить содержимое ячейки памяти с *A*	1	0	0	0	0	1	1	0	7
ADDr	Сложить содержимое регистра с *A*	1	0	0	0	0	*S*	*S*	*S*	4
ADI	Сложить непосредственно с *A*	1	1	0	0	0	1	1	0	7
ANAM	Логическое *И* содержимого ячейки памяти с *A*	1	0	1	0	0	1	1	0	7
ANAr	Логическое *И* содержимого регистра с *A*	1	0	1	0	0	*S*	*S*	*S*	4
ANI	Логическое *И* непосредственно с *A*	1	1	1	0	0	1	1	0	7
CALL	Вызов подпрограммы	1	1	0	0	1	1	0	1	17
CC	Вызов по переносу	1	1	0	1	1	1	0	0	11/17
CM	Вызов по минусу	1	1	1	1	1	1	0	0	11/17
CMA	Дополнение *A*	0	0	1	0	1	1	1	1	4
CMC	Дополнение переноса	0	0	1	1	1	1	1	1	4
CMP M	Сравнить содержимое ячейки памяти с *A*	1	0	1	1	1	1	1	0	7
CMP r	Сравнить содержимое регистра с *A*	1	0	1	1	1	*S*	*S*	*S*	4
CNC	Вызов по отсутствию переноса	1	1	0	1	0	1	0	0	11/17
CNZ	Вызов по не нулю	1	1	0	0	0	1	0	0	11/17
CP	Вызов по плюсу	1	1	1	1	0	1	0	0	11/17
CPE	Вызов по четности	1	1	1	0	1	1	0	0	11/17
CPI	Сравнить непосредственно с *A*	1	1	1	1	1	1	1	0	7
CPO	Вызов по нечетности	1	1	1	0	0	1	0	0	11/17
CZ	Вызов по нулю	1	1	0	0	1	1	0	0	11/17
DAA	Десятичная коррекция *A*	0	0	1	0	0	1	1	1	4
DADB	Сложить пару *BC* с *HL*	0	0	0	0	1	0	0	1	10
DAD D	Сложить пару *DE* с *HL*	0	0	0	1	1	0	0	1	10
DAD H	Сложить пару *HL* с *HL*	0	0	1	0	1	0	0	1	10
DAD SP	Сложить указатель стека с *HL*	0	0	1	1	1	0	0	1	10
DCR M	Декремент содержимого ячейки памяти	0	0	1	1	0	1	0	1	10
DCR r	Декремент содержимого регистра	0	0	*D*	*D*	*D*	1	0	1	5
DCX B	Декремент *BC*	0	0	0	0	1	0	1	1	5
DCX D	Декремент *DE*	0	0	0	1	1	0	1	1	5
DCX H	Декремент *HL*	0	0	1	0	1	0	1	1	5
DCX SP	Декремент указателя стека	0	0	1	1	1	0	1	1	5
DI	Запрет прерываний	1	1	1	1	0	0	1	1	4
EI	Разрешение прерываний	1	1	1	1	1	0	1	1	4
HLT	Останов	0	1	1	1	0	1	1	0	7
IN	Ввод	1	1	0	1	1	0	1	1	10
INRM	Инкремент содержимого ячейки памяти	0	0	1	1	0	1	0	0	10
INR r	Инкремент содержимого регистра	0	0	*D*	*D*	*D*	1	0	0	5
INX B	Инкремент пары *BC*	0	0	0	0	0	0	1	1	5
INX D	Инкремент пары *DE*	0	0	0	1	0	0	1	1	5
INX H	Инкремент пары *HL*	0	0	1	0	0	0	1	1	5
INX SP	Инкремент указателя стека	0	0	1	1	0	0	1	1	5
JC	Переход по переносу	1	1	0	1	1	0	1	0	10
JM	Переход по минусу	1	1	1	1	1	0	1	0	10
JMP	Безусловный переход	1	1	0	0	0	0	1	1	10
JNC	Переход по отсутствию переноса	1	1	0	1	0	0	1	0	10
JNZ	Переход по не нулю	1	1	0	0	0	0	1	0	10
JP	Переход по плюсу	1	1	1	1	0	0	1	0	10

Мнемоника	Описание	Код операции (1) D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Длительность в циклах (2)
JPE	Переход по четности	1	1	1	0	1	0	1	0	10
JPO	Переход по нечетности	1	1	1	0	0	0	1	0	10
JZ	Переход по нулю	1	1	0	0	1	0	1	0	10
LDA	Прямая загрузка *A*	0	0	1	1	1	0	1	0	13
LDAX B	Косвенная загрузка *A* по *BC*	0	0	0	0	1	0	1	0	7
LDAX D	Косвенная загрузка *A* по *DE*	0	0	0	1	1	0	1	0	7
LHL D	Прямая загрузка *HL*	0	0	1	0	1	0	1	0	16
LXI B	Непосредственная загрузка пары *BC*	0	0	0	0	0	0	0	1	10
LXI D	Непосредственная загрузка пары *DE*	0	0	0	1	0	0	0	1	10
LXI H	Непосредственная загрузка пары *HL*	0	0	1	0	0	0	0	1	10
LXI SP	Непосредственная загрузка указателя стека	0	0	1	1	0	0	0	1	10
MVI M	Непосредственная загрузка ячейки памяти	0	0	1	1	0	1	1	0	10
MVI r	Непосредственная загрузка регистра	0	0	*D*	*D*	*D*	1	1	0	7
MOV M, r	Пересылка содержимого регистра в ячейку памяти	0	1	1	1	0	*S*	*S*	*S*	7
MOV r, M	Пересылка содержимого ячейки памяти в регистр	0	1	*D*	*D*	*D*	1	1	0	7
MOV r1,r2	Пересылка содержимого регистра в регистр	0	1	*D*	*D*	*D*	*S*	*S*	*S*	5
NOP	Нет операций	0	0	0	0	0	0	0	0	4
ORA M	Логическое *ИЛИ* содержимого ячейки памяти с *A*	1	0	1	1	0	1	1	0	7
ORA r	Логическое *ИЛИ* содержимого регистра с *A*	1	0	1	1	0	*S*	*S*	*S*	4
ORI	Логическое *ИЛИ* непосредственно с *A*	1	1	1	1	0	1	1	0	7
OUT	Вывод	1	1	0	1	0	0	1	1	10
PCHL	Пересылка содержимого *HL* в программный счетчик	1	1	1	0	1	0	0	1	5
POP B	Извлечение содержимого пары регистров *BC* из стека	1	1	0	0	0	0	0	1	10
POP D	Извлечение содержимого пары регистров *DE* из стека	1	1	0	1	0	0	0	1	10
POP H	Извлечение содержимого пары регистров *HL* из стека	1	1	1	0	0	0	0	1	10
POP PSW	Извлечение содержимого регистра *A* и флажков из стека	1	1	1	1	0	0	0	1	10
PUSH B	Засылка содержимого пары регистров *BC* в стек	1	1	0	0	0	1	0	1	11
PUSH D	Засылка содержимого пары регистров *DE* в стек	1	1	0	1	0	1	0	1	11
PUSH H	Засылка содержимого пары регистров *HL* в стек	1	1	1	0	0	1	0	1	11
PUSH PSW	Засылка содержимого регистра *A* и флажков в стек	1	1	1	1	0	1	0	1	11
RAL	Циклический сдвиг *A* влево через перенос	0	0	0	1	0	1	1	1	4

Продолжение рис. 3.28

Мнемоника	Описание	Код операции (1) D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Длительность в циклах (2)
RAR	Циклический сдвиг *A* вправо через перенос	0	0	0	1	1	1	1	1	4
RC	Возврат по переносу	1	1	0	1	1	0	0	0	5/11
RET	Возврат из подпрограммы	1	1	0	0	1	0	0	1	10
RLC	Циклический сдвиг *A* влево	0	0	0	0	0	1	1	1	4
RM	Возврат по минусу	1	1	1	1	1	0	0	0	5/11
RNC	Возврат по отсутствию переноса	1	1	0	1	0	0	0	0	5/11
RNZ	Возврат по не нулю	1	1	0	0	0	0	0	0	5/11
RP	Возврат по плюсу	1	1	1	1	0	0	0	0	5/11
RPE	Возврат по четности	1	1	1	0	1	0	0	0	5/11
RPO	Возврат по нечетности	1	1	1	0	0	0	0	0	5/11
RRC	Циклический сдвиг *A* вправо	0	0	0	0	1	1	1	1	4
RST	Рестарт	1	1	*A*	*A*	*A*	1	1	1	11
RZ	Возврат по нулю	1	1	0	0	1	0	0	0	5/11
SBB M	Вычесть содержимое ячейки памяти из *A* с заемом	1	0	0	1	1	1	1	0	7
SBB r	Вычесть содержимое регистра из *A* с заемом	1	0	0	1	1	*S*	*S*	*S*	4
SBI	Вычесть непосредственно из *A* с заемом	1	1	0	1	1	1	1	0	7
SHLD	Запомнить прямо *HL*	0	0	1	0	0	0	1	0	16
SPHL	Переслать содержимое *HL* в указатель стека	1	1	1	1	1	0	0	1	5
STA	Запомнить прямо *A*	0	0	1	1	0	0	1	0	13
STAX B	Запомнить косвенно *A* по *BC*	0	0	0	0	0	0	1	0	7
STAX D	Запомнить косвенно *A* по *DE*	0	0	0	1	0	0	1	0	7
STC	Установить перенос	0	0	1	1	0	1	1	1	4
SUBM	Вычесть содержимое ячейки памяти из *A*	1	0	0	1	0	1	1	0	7
SUB r	Вычесть содержимое регистра из A	1	0	0	1	0	*S*	*S*	*S*	4
SUI	Вычесть непосредственно из *A*	1	1	0	1	0	1	1	0	7
XCHG	Обменять содержимое пар *DE* и *HL*	1	1	1	0	1	0	1	1	4
XRAM	Исключающее *ИЛИ* содержимого ячейки памяти с *A*	1	0	1	0	1	1	1	0	7
XRA r	Исключающее *ИЛИ* содержимого регистра с *A*	1	0	1	0	1	*S*	*S*	*S*	4
XRI	Исключающее *ИЛИ* непосредственно с *A*	1	1	1	0	1	1	1	0	7
XTHL	Обмен содержимого верхушки стека с *HL*	1	1	1	0	0	0	1	1	18

Примечания: 1. *DDD* или *SSS*: 000 — *B*, 001 — *C*, 010 — *D*, 011 — *E*, 100 — *H*, 101 — *L*, 110 — память, 111 — *A*.
2. Две возможных длительности команды указывают на зависимость числа циклов от состояния флажков.

Рис. 3.28. Система команд *INTEL 8080*

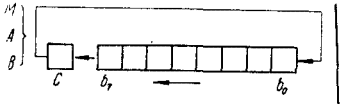
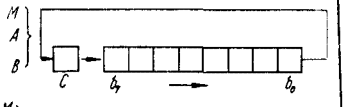
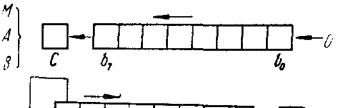
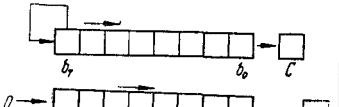
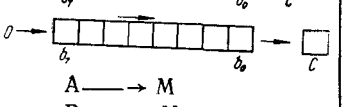
Команды обращения к памяти и аккумуляторам	Мнемо-код	Адресация								
		Непосредственная			Прямая			Индексная		
		OP	~	#	OP	~	#	OP	~	#
Сложить	ADDA	8B	2	2	9B	3	2	AB	5	2
	ADDB	CB	2	2	DB	3	2	EB	5	2
Сложить аккумуляторы	ABA									
Сложить с переносом	ADCA	89	2	2	99	3	2	A9	5	2
	ADCB	C9	2	2	D9	3	2	E9	5	2
Логическое *И*	ANDA	84	2	2	94	3	2	A4	5	2
	ANDB	C4	2	2	D4	3	2	E4	5	2
Побитовая проверка	BITA	85	2	2	95	3	2	A5	5	2
	BITB	C5	2	2	D5	3	2	E5	5	2
Очистить	CLR							6F	7	2
	CLRA									
	CLRB									
Сравнить	CMPA	81	2	2	91	3	2	A1	5	2
	CMPB	C1	2	2	D1	3	2	E1	5	2
Сравнить аккумуляторы	CBA									
Образовать обратный код	COM							63	7	2
	COMA									
	COMB									
Образовать дополнительный код	NEG							60	7	2
	NEGA									
	NEGB									
Десятичная коррекция, *А*	DAA									
Уменьшить на единицу	DEC							6A	7	2
	DECA									
	DECB									
Исключающее *ИЛИ*	EORA	88	2	2	98	3	2	A8	5	2
	EORB	C8	2	2	D8	3	2	E8	5	2
Увеличить на единицу	INC							6C	7	2
	INCA									
	INCB									
Загрузить аккумуляторы	LDAA	86	2	2	96	3	2	A6	5	2
	LDAB	C6	2	2	D6	3	2	E6	5	2
Логическое *ИЛИ*	ORAA	8A	2	2	9A	3	2	AA	5	2
	ORAB	CA	2	2	DA	3	2	EA	5	2
Засылка в стек	PSHA									
	PSHB									
Извлечение из стека	PULA									
	PULB									
Циклический сдвиг влево	ROL							69	7	2

Расширенная			Безадресная			Мнемоника выполнения (все наименования регистров обозначают их содержимое)	Регистр кода условия					
							5	4	3	2	1	0
OP	⌣	#	OP	⌣	#		H	I	N	Z	V	C
BB	4	3				A + M ⟶ A	↕	·	↕	↕	↕	↕
FB	4	3				B + M ⟶ B	↕	·	↕	↕	↕	↕
			7B	2	1	A + B ⟶ A	↕	·	↕	↕	↕	↕
B9	4	3				A + M + C ⟶ A	↕	·	↕	↕	↕	↕
F9	4	3				B + M + C ⟶ B	↕	·	↕	↕	↕	↕
B4	4	3				A · M ⟶ A	·	·	↕	↕	R	·
F4	4	3				B · M ⟶ B	·	·	↕	↕	R	·
B5	4	3				A · M	·	·	↕	↕	R	·
F5	4	3				B · M	·	·	↕	↕	R	·
7F	6	3				00 ⟶ M	·	·	R	S	R	R
			4F	2	1	00 ⟶ A	·	·	R	S	R	R
			5F	2	1	00 ⟶ B	·	·	R	S	R	R
B1	4	3				A — M	·	·	↕	↕	↕	↕
F1	4	3				B — M	·	·	↕	↕	↕	↕
			11	2	1	A — B	·	·	↕	↕	↕	↕
73	6	3				$\overline{M}$ ⟶ M	·	·	↕	↕	R	S
			43	2	1	$\overline{A}$ ⟶ A	·	·	↕	↕	R	S
			53	2	1	$\overline{B}$ ⟶ B	·	·	↕	↕	R	S
70	6	3				00 — M ⟶ M	·	·	↕	↕	1	2
			40	2	1	00 — A ⟶ A	·	·	↕	↕	1	2
			50	2	1	00 — B ⟶ B	·	·	↕	↕	1	2
			19	2	1	Преобразование двоичной суммы двоично-десятичных символов в двоично-десятичный код	·	·	↕	↕	↕	3
7A	6	3				M — 1 ⟶ M	·	·	↕	↕	4	·
			4A	2	1	A — 1 ⟶ A	·	·	↕	↑	4	·
			5A	2	1	B — 1 ⟶ B	·	·	↕	↑	4	·
B8	4	3				A ⊕ M ⟶ A	·	·	↕	↕	R	·
F8	4	3				B ⊕ M ⟶ B	·	·	↕	↕	R	·
7C	6	3				M + 1 ⟶ M	·	·	↕	↕	5	·
			4C	2	1	A + 1 ⟶ A	·	·	↕	↕	5	·
			5C	2	1	B + 1 ⟶ B	·	·	↕	↕	5	·
B6	4	3				M ⟶ A	·	·	↕	↕	R	·
F6	4	3				M ⟶ B	·	·	↕	↕	R	·
BA	4	3				A + M ⟶ A	·	·	↕	↕	R	·
FA	4	3				B + M ⟶ B	·	·	↕	↕	R	·
			36	4	1	A ⟶ M_{SP}, SP—1 ⟶ SP	·	·	·	·	·	·
			37	4	1	B ⟶ M_{SP}, SP—1 ⟶ SP	·	·	·	·	·	·
			32	4	1	SP + 1 ⟶ SP, M_{SP} ⟶ A	·	·	·	·	·	·
			33	4	1	SP + 1 ⟶ SP, M_{SP} ⟶ B	·	·	·	·	·	·
79	6	3					·	·	↕	↕	6	↕

Команды обращения к памяти и аккумуляторам	Мнемокод	Адресация								
		Непосредственная			Прямая			Индексная		
		OP	~	#	OP	~	#	OP	~	#
	ROLA									
	ROLB									
Циклический сдвиг вправо	ROR							66	7	2
	RORA									
	RORB									
Арифметический сдвиг влево	ASL							68	7	2
	ASLA									
	ASLB									
Арифметический сдвиг вправо	ASR							67	7	2
	ASRA									
	ASRB									
Логический сдвиг вправо	LSR							64	7	2
	LSRA									
	LSRB									
Запомнить содержимое аккумулятора	STAA				97	4	2	A7	6	2
	STAB				D7	4	2	E7	6	2
Вычесть	SUBA	80	2	2	90	3	2	A0	5	2
	SUBB	C0	2	2	D0	3	2	E0	5	2
Вычесть аккумуляторы	SBA									
Вычесть с переносом	SBCA	82	2	2	92	3	2	A2	5	2
	SBCB	C2	2	2	D2	3	2	E2	5	2
Обмен содержимым аккумуляторов	TAB									
	TBA									
Проверка на нуль и отрицательное число	TST							60	7	2
	TSTA									
	TSTB									

Команды управления стеком и

Сравнить с индексным регистром	CPX	8C	3	3	9C	4	2	AC	6	2
Уменьшить индексный регистр	DEX									
Уменьшить указатель стека	DES									
Увеличить индексный регистр	INX									
Увеличить указатель стека	INS									

Расширенная			Безадресная			Мнемоника выполнения (все наименования регистров обозначают их содержимое)	Регистр кода условия					
							5	4	3	2	1	0
OP	~	#	OP	~	#		H	I	N	Z	V	C
			49	2	1		·	·	↕	↕	6	↕
			59	2	1		·	·	↕	↕	6	↕
76	6	3					·	·	↕	↕	6	↕
			46	2	1		·	·	↕	↕	6	↕
			56	2	1		·	·	↑	↑	6	↕
78	6	3					·	·	↕	↕	6	↕
			48	2	1		·	·	↑	↑	6	↕
			58	2	1		·	·	↕	↕	6	↕
77	6	3					·	·	↕	↕	6	↕
			47	2	1		·	·	↕	↕	6	↕
			57	2	1		·	·	↕	↕	6	↕
74	6	3					·	·	R	↕	6	↕
			44	2	1		·	·	R	↕	6	↕
			54	2	1		·	·	R	↕	6	↕
B7	5	3				A ——→ M	·	·	↕	↕	R	↕
F7	5	3				B ——→ M	·	·	↕	↕	R	·
B0	4	3				A — M ——→ A	·	·	↕	↕	↕	↕
F0	4	3				B — M ——→ B	·	·	↕	↕	↕	↕
			10	2	1	A — B ——→ A	·	·	↕	↕	↕	↕
B2	4	3				A — M — C ——→ A	·	·	↕	↕	↕	↕
F2	4	3				B — M —C ——→ B	·	·	↕	↕	↕	↕
			16	2	1	A ——→ B	·	·	↕	↕	R	·
			17	2	1	B ——→ A	·	·	↕	↕	R	·
70	6	3				M — 00	·	·	↕	↕	R	R
			40	2	1	A — 00	·	·	↕	↕	R	R
			50	2	1	B — 00	·	·	↕	↕	R	R

индексным регистром

BC	5	3				X_H — M, X_L — (M + 1)	·	·	7	↕	8	·
			09	4	1	X — 1 ——→ X	·	·	·	↕	·	·
			34	4	1	SP — 1 ——→ SP	·	·	·	·	·	·
			08	4	1	X + 1 ——→ X	·	·	·	↕	·	·
			31	4	1	SP + 1 ——→ SP	·	·	·	·	·	·

Команды обращения к памяти и аккумуляторам	Мнемокод	Адресация								
		Непосредственная			Прямая			Индексная		
		OP	~	#	OP	~	#	OP	~	#
Загрузить индексный регистр	LDX	CE	3	3	DE	4	2	EE	6	2
Загрузить указатель стека	LDS	8E	3	3	9E	4	2	AE	6	2
Запомнить индексный регистр	STX				DF	5	2	EF	7	2
Запомнить указатель стека	STS				9F	5	2	AF	7	2
Переслать из индексного регистра в указатель стека	TXS									
Переслать из указателя стека в индексный регистр	TSX									

Команды передачи управления и ветвления программ	Мнемокод	Адре					
		Относительная			Индексная		
		OP	~	#	OP	~	#
Перейти при любых условиях	BRA	20	4	2			
Перейти по отсутствию переноса	BCC	24	4	2			
Перейти по наличию переноса	BCS	25	4	2			
Перейти по условию $A = 0$	BEQ	27	4	2			
Перейти по условию $A \geqslant 0$	BGE	2C	4	2			
Перейти по условию $A > 0$	BGT	2E	4	2			
Перейти по условию, если больше	BHI	22	4	2			
Перейти по условию $A \leqslant 0$	BLE	2F	4	2			
Перейти по условию, если меньше или равно	BLS	23	4	2			
Перейти по условию $A < 0$	BLT	2D	4	2			
Перейти по отрицательному числу	BMI	2B	4	2			
Перейти по условию $A \neq 0$	BNE	26	4	2			
Перейти по отсутствию переполнения	BVC	28	4	2			
Перейти по наличию переполнения	BVS	29	4	2			
Перейти по положительному числу	BPL	2A	4	2			
Перейти на подпрограмму	BSR	8D	8	2			
Безусловный переход	JMP				6E	4	2
Обращение к подпрограмме	JSR				AD	8	2
Нет операции	NOP						
Возврат после прерывания	RTI						
Возврат из подпрограммы	RTS						
Программное прерывание	SWI						
Ожидать прерывание	WAI						

WAI устанавливает адресную шину, запись/чтение и шину данных в трехстабиль-

Расширенная			Безадресная			Мнемоника выполнения (все наименования регистров обозначают их содержимое)	Регистр кода условия					
							5	4	3	2	1	0
OP	~	#	OP	~	#		H	I	N	Z	V	C
FE	5	3				$M \longrightarrow X_H$, $(M+1) \longrightarrow X_L$	·	·	9	↕	R	·
BE	5	3				$M \longrightarrow SP_H$, $(M+1) \longrightarrow SP_L$	·	·	9	↕	R	·
FF	6	3				$X_H \longrightarrow M$, $X_L \longrightarrow (M+1)$	·	·	9	↕	R	·
BF	6	3				$SP_H \longrightarrow M$, $SP_L \longrightarrow (M+1)$	·	·	9	↕	R	·
			35	4	1	$X - 1 \longrightarrow SP$	·	·	·	·	·	·
			30	4	1	$SP + 1 \longrightarrow X$	·	·	·	·	·	·

сация													
Расширенная			Безадресная			Проверяемое условие перехода	Регистр кода состояния						
							5	4	3	2	1	0	
OP	~	#	OP	~	#		H	I	N	Z	V	C	
						Не проверяется	·		·	·	·	·	
						C = 0	·	·	·	·	·	·	
						C = 1	·	·	·	·	·	·	
						Z = 1	·	·	·	·	·	·	
						$N \oplus V = 0$	·	·	·	·	·	·	
						$Z + (N \oplus V = 0$	·	·	·	·	·	·	
						$C + Z = 0$	·	·	·	·	·	·	
						$Z + (N \oplus V) = 1$	·	·	·	·	·	·	
						$C + Z = 1$	·	·	·	·	·	·	
						$N \oplus V = 1$	·	·	·	·	·	·	
						N = 1	·	·	·	·	·	·	
						Z = 0	·	·	·	·	·	·	
						V = 0	·	·	·	·	·	·	
						V = 1	·	·	·	·	·	·	
						N = 0	·	·	·	·	·	·	
						Смотри специальные команды	·	·	·	·	·	·	
7E	3	3				Смотри специальные команды	·	·	·	·	·	·	
B	9	3											
			01	2	1	Только увеличивается программный счетчик	·	·	·	·	·	·	
			3B	10	1		10						
			39	5	1	Смотри специальные команды	·	·	·	·	·	·	
			3F	12	1		·	S	·	·	·	·	
			3E	9	1		·	11	·	·	·	·	

ное состояние, когда сигнал VMA принимает значение «0».

Продолжение рис. 3.30

Команды управления регистром кода состояния	Мнемокод	Безадресная			Мнемоника	Регистр кода состояния					
		OP	~	#		5	4	3	2	1	0
						H	I	N	Z	V	C
Очистить перенос	CLC	OC	2	1	0→C	·	·	·	·	·	R
Сбросить маску прерывания	CLI	OE	2	1	0→I	·	R	·	·	·	·
Очистить переполнение	CLV	OA	2	1	0→V	·	·	·	·	R	·
Установить перенос	SEC	OD	2	1	1→C	·	·	·	·	·	S
Установить маску прерывания	SEI	OF	2	1	1→I	·	S	·	·	·	·
Установить переполнение	SEV	OB	2	1	1→V	·	·	·	·	S	·
Переслать из аккумулятора в регистр состояния	TAP	06	2	1	A→CCR	12					
Переслать из регистра состояния в аккумулятор	TPA	07	2	1	CCR→A	·	·	·	·	·	·

Описание назначения разрядов кода состояния
и условий их установки

(Бит устанавливается в единицу, если проверяемое условие истинно, и сбрасывается в противном случае)

1 (Бит V) Проверка: Результат =10000000?

2 (Бит C) Проверка: Результат =00000000?

3 (Бит C) Проверка: Десятичное значение старшего символа двоично-десятичного кода больше девяти? (не сбрасывается, если был предварительно установлен)

4 (Бит V) Проверка: Операнд =10000000 перед выполнением команды?

5 (Бит V) Проверка: Операнд =01111111 перед выполнением команды?

6 (Бит V) Проверка: Устанавливается равным N⊕C после выполнения сдвига

7 (Бит N) Проверка: Знаковый разряд старшего байта =1?

8 (Бит N) Проверка: Имеется ли переполнение в дополнительном коде при вычитании старшего байта?

9 (Бит N) Проверка: Результат меньше нуля? (15-й бит =1)

10 (Все) Загрузить регистр состояния из стека (смотри специальные команды)

11 (Бит I) Устанавливается, когда возникает прерывание. Если предварительно был установлен, требуется немаскируемое прерывание для выхода из состояния ожидания

12 (Все) Устанавливаются в соответствии с содержимым аккумулятора

Продолжение рис. 3.30

Условные обозначения:

OP — Код операции
~ — Число циклов ЦП
# — Число байтов
+ — Плюс арифметический
— — Минус арифметический
· — Логическое *И*
MSP — Содержимое ячейки памяти, указываемой указателем стека
∨ — Логическое *ИЛИ*
⊕ — Исключающее *ИЛИ*
$\overline{M}$ — Инверсия *M*
→ — Пересылка в
0 — Нулевой бит
00 — Нулевой байт
H — Дополнительный перенос из четвертого разряда (бит 3)
I — Маска прерывания
N — Отрицательно (знаковый разряд)
Z — Нуль (байт)
V — Переполнение, признак дополнительного кода
C — Перенос из 8 разряда (бит 7)
R — Сброс
S — Установка
↕ — Проверяется и устанавливается, если истинно, сбрасывается в противном случае
● — Не изменяется командой

Рис. 3.30. Система команд *Motorola 6800*

Шаг 5 — проектирование программного обеспечения. Восьмеричный и шестнадцатиричный листинги для МП *In 8080* составляются непосредственно по схеме алгоритма (рис. 3.25), используя карту распределения команд на рис. 3.27 и систему команд на рис. 3.28.

Решение для *Mc6800.*

Шаг 1
Шаг 2 } То же самое, что и для МП *In 8080.*
Шаг 3

Шаг 4 — проектирование аппаратурной части. Структурная схема устройства на базе *Mc6800* показана на рис. 3.29. В дополнение к порту ввода/вывода требуется схема *И* для формирования сигнала разрешения *e,* как объяснено в 3.4 (см. схемы 1 и 2).

Шаг 5 — проектирование программного обеспечения. Из структурной схемы алгоритма (рис. 3.25), системы команд (рис. 3.30) или карты распределения команд (рис. 3.31) непосредственно составляется шестнадцатиричный листинг программы (табл. 3.2).

Задача 2. Чтение и распечатка *n* символов. Имеется микропроцессор, считыватель с перфоленты и печатающее устройство. Необходимо спроектировать систему, печатающую *n* первых символов, введенных с перфоленты.

Для реализации системы использовать логическую схему ожидания/счета МП *In 8080* и *Mc6800.*

Решение для *In 8080.*

Шаг 1 — формирование технического задания. Считывание и распечатка *n* символов.

Шаг 2 — внешнее проектирование. Микропроцессор имеет логическую схему ожидания/счета. Внешние устройства относятся к типу запрос/ответ (см. приложение 1).

		0						D_6
		$D_2D_1D_0$						
		000	010	110	100	101	111	011
0	000			$CC:=A$			$A:=CC$	
	010	$A:=A-B$		$B:=A$			$A:=B$	
	110	$IX:=SP+1$	$POP\ A$	$PUSH\ A$	$SP:=SP-1$	$SP:=IX-1$	$PUSH\ B$	$POP\ B$
	100	$JUMP$	$\overline{cz}$	$\bar{z}$	$\bar{c}$	c		$c+z$
	$D_5D_4D_3$	Условный переход						
	101	$\bar{v}$	$\bar{n}$	$\bar{z}$ $nv+\overline{nv}$	$nv+\overline{nv}$	$n\bar{v}+\bar{n}v$	$z+n\bar{v}+\bar{n}v$	n
D_7	111			$WAIT$			$INT\ SOFT$	RTI
	011							$A:=A+B$
	001	$IX:=IX+1$	$v:=0$	$I:=0$	$c:=0$	$c:=1$	$I:=1$	$v:=1$
1	001					Вызов		
	011	$A:=A\vee p$	$A:=A\vee p$	$SP:=p$	$TEST\ (IX-p)$		$p=SP$	$A:=A+p$
	111							
	101					Вызов		
	$D_5D_4D_3$							
	100							
	110	$A:=A-p$	$A:=A-p-c$	$A:=p$	$A:=A\wedge p$	$TEST\ (A\wedge p)$	$p:=A$	
	010							
	000							

Операнд

D_5D_4	p (режим адресации)	g (содержимое регистра или ячейки памяти)
00	*IMM* (непосредственно)	A
01	*DIR* (прямо)	B
11	*EXT* (расширено)	*EXT*
10	*IND* (индексно)	*IND* при $D_4=1$ или *REL* при $D_6=0$ (относительно)

Рис. 3.31. Карта команд *Motorola 6800*

			$D_2D_1D_0$					
001	001	011	111	101	100	110	010	000
NOP		$g:=\bar{g}$	Арифметический сдвиг вправо		Логический сдвиг вправо	Циклический сдвиг вправо		
Сравнить *A*, *B*			*g*		*g*	*g*		$g:=-g$
$SP:=SP+1$								
v	Циклический сдвиг влево					*JMP IND*		Сдвиг влево
RTS			$g:=0$	*TEST* $(g-0)$	$g:=g+1$	*JMP EXT*	$g:=g-1$	*g*
DAA	*g*							
$IX:=IX-1$								
$A:=$ $=A+p+c$	$B:=$ $=B+p+c$	$B:=B+p$	$p:=IX$			$IX:=p$	$B:=B\forall p$	$B:=B\vee p$
TEST $(A-p)$	*TEST* $(B-p)$		$p:=B$	*TEST* $(B\wedge p)$	$B:=B\wedge p$	$B:=p$	$B:=$ $=B-p-c$	$B:=B-p$

Регистры

A аккумулятор (8)

B аккумулятор (8)

IX индексный регистр (16)

PC программный счетчик (16)

SP указатель стека (16)

CC биты состояния (6)

Биты состояния

c перенос-заем

v переполнение

$\bar{z}$ нуль

n отрицательно (знак)

i маска прерываний

h промежуточный перенос

Таблица 3.2

Программа поиска записи для МП *Мс6800*

	16-й адрес		16-й код	Мнемокод	Комментарии
	H	L			
L1:	00	00	B6	LDA A, 2000	Читать символ
		01	20		
		02	00		
		03	01	NOP	
L2:		04	81	CMP A	Сравнить содержимое *A* с 4
		05	34		
		06	26	BNE L1	Если символ не равен 4, перейти к *L1*
		07	F8		
		08	B6	LDA A, 2000	Читать символ
		09	20		
		OA	00		
		OB	01	NOP	
		OC	81	CMP A	Сравнить содержимое *A* с 5
		OD	35		
		OE	26	BNE L2	Если символ не равен 5, перейти к *L2*
		OF	F4		
		10	B6	LDA A, 2000	Читать символ
		11	20		
		12	00		
		13	01	NOP	
		14	81	CMP A	Сравнить содержимое *A* с 6
		15	36		
		16	26	BNE L2	Если символ не равен 6, перейти к 2
		17	EC		
		18	B7	STA A, 4000	Установить флажок
		19	40		
		1A	00		
		1B	3F	SWI	Останов

Шаг 3 — системное проектирование. Структурная схема проектируемой системы изображена на рис. 3.32. Так как печатаю-

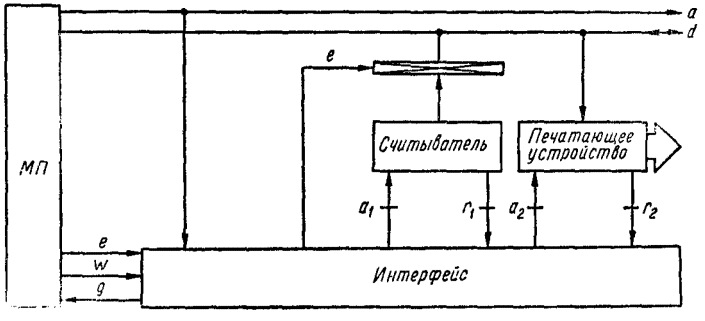


Рис. 3.32

щее устройство является приемником данных, то для него не нужен порт ввода/вывода. Структурная схема алгоритма функционирования системы показана на рис. 3.33.

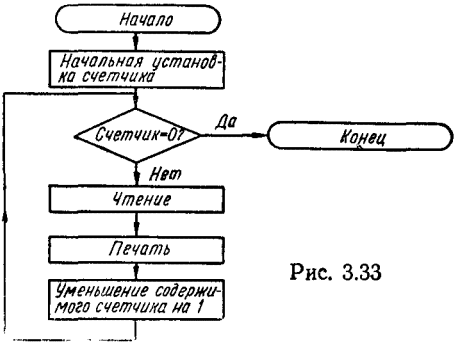


Рис. 3.33

Шаг 4 — проектирование аппаратурной части. За исключением порта ввода/вывода для фотосчитывателя (рис. 3.34) другие аппаратурные средства не требуются. Это связано с тем, что для МП *In 8080* $e=w$ (см. пример 14 в параграфе 3.4).

Шаг 5 — проектирование программного обеспечения. Восьмеричный и шестнадцатиричный листинги (табл. 3.3) программы для МП *In 8080* получены непосредственно из схемы алгоритма (рис. 3.33) с использованием системы команд (рис. 3.27 и 3.28).

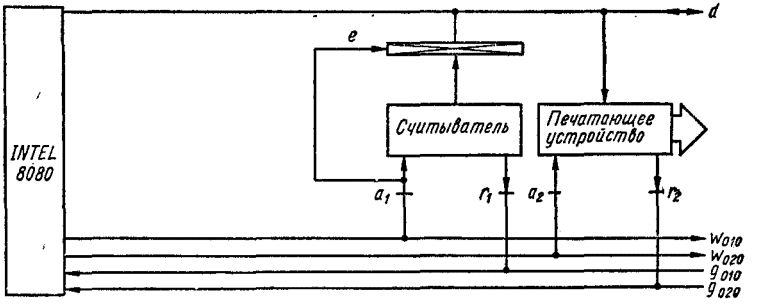


Рис. 3.34

Решение для *Mc6800.*

Шаг 1
Шаг 2 } То же самое, что и для МП *In 8080.*
Шаг 3

Шаг 4 — проектирование аппаратурной части. Структурная схема системы на базе *Mc6800* показана на рис. 3.35.

Как и в предыдущей задаче, в дополнение к порту ввода/вывода требуется схема *И* для формирования сигнала разрешения *e* (см. пример 15 в параграфе 3.4 схемы 1 и 2).

Шаг 5 — проектирование программного обеспечения. Непосредственно по схеме алгоритма с использованием набора команд *Mc6800* (рис. 3.30 и 3.31) получим листинг программы (табл. 3.4).

Задача 3. Распечатать запись. Имеются считыватель с перфоленты, печатающее устройство и микропроцессор. Необходимо спроектировать систему, печатающую запись из n символов.

Таблица 3.3

Программа распечатки *n* символов со считывателя для *In8080*

	8-й адрес		8-й код	16-й код	Мнемо-код	Комментарии	
	H	L					
	003	000	016	OE	MVI C	Загрузить число в регистр C	
		001	(*n*)	(*n*)			
		002	014	OC	INR C	Увеличить содержимое регистра *C*	Установить флажки условий
L2:		003	015	OD	DCR C	Уменьшить содержимое регистра *C*	
		004	312	CA	JZ	Если содержимое регистра *C*=0,	
		005	016	OE	L1	перейти к L1	
		006	003	03			
		007	333	DB	IN	Читать следующий символ	
		010	010	08			
		011	323	D3	OUT	Печатать символ	
		012	020	18			
		013	303	C3	JMP	Перейти к L2	
		014	003	03	L2		
		015	003	03			
L1:		016	166	76	HLT	Останов	

Метка записи — последовательность символов 4—5—6, которая применяется только для этих целей.

Для построения системы использовать режим ожидания/счета.

Решение для *In8080*.

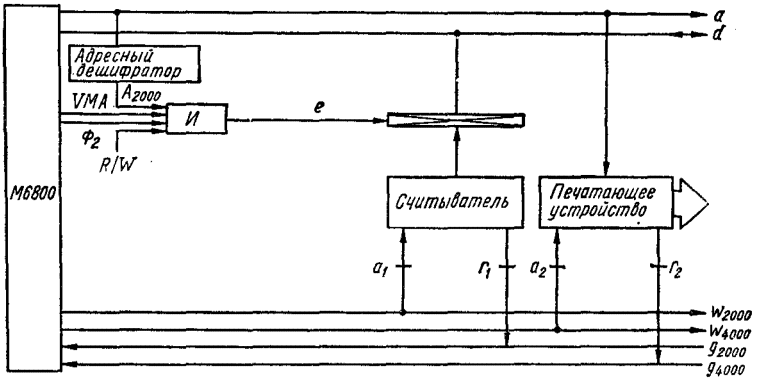


Рис. 3.35

Шаг 1 — формирование технического задания. Целью проектирования является создание устройства, способного распечатывать запись.

Шаг 2 — внешнее проектирование. Микропроцессор располагает логической схемой ожидания/счета. Фотосчитыватель и пе-

Таблица 3.4

Программа распечатки *n* символов со считывателя для *Mc6800*

	16-й адрес		16-й код	Мнемокод	Комментарии
	H	L			
	00	00	C6	LDA B, *n*	Загрузить в регистр B
		01	*n*	*n*	
L2:		02	27	BEQ	Если содержимое регистра B
		03	0B	L1	равно 0, перейти к L1
		04	B6	LDA A, 2000	Читать символ
		05	20		
		06	00		
		07	01	NOP	
		08	B7	STA A, 4000	Печатать символ
		09	40		
		0A	00		
		0B	01	NOP	
		0C	5A	DEC B	Уменьшить содержимое B
		0D	7E	JMP L2	Перейти к L2
		0E	00		
		0F	02		
L1:		10	3F	SWI	Останов

чатающее устройство являются устройствами типа запрос/ответ (см. приложение 1).

Шаг 3 — системное проектирование. Структурная схема разрабатываемой системы показана на рис. 3.32, а структурная схема алгоритма его функционирования — на рис. 3.36.

Шаг 4 — проектирование аппаратурной части. За исключением порта ввода/вывода (рис. 3.34) другие аппаратурные средства не требуются, так как для *In 8080* $e=w$ (см. пример 14 в параграфе 3. 4).

Шаг 5 — проектирование программного обеспечения. Непосредственно из схемы алгоритма (рис. 3.36) и набора ко-

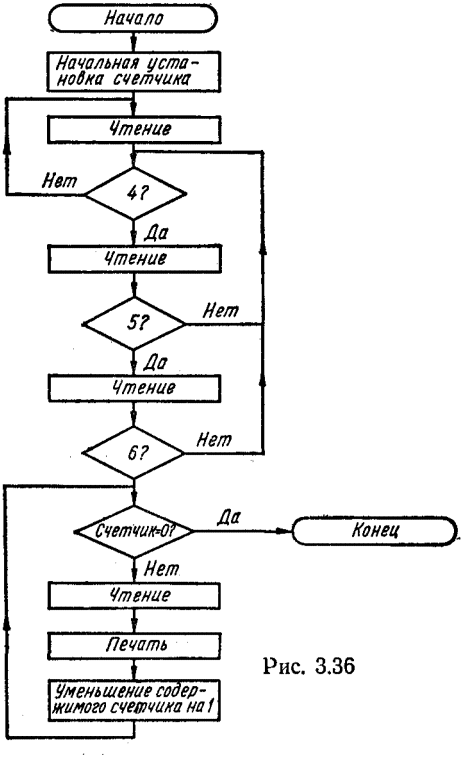


Рис. 3.36

Таблица 3.5

Программа распечатки записи со считывателя по метке для *In8080*

	16-й адрес		8-й код	16-й код	Мнемокод	Комментарии
	H	L				
L1:	003	000	333	DB	IN	Читать символ
		001	020	10		
L2:		002	376	FE	CPI	Сравнить содержимое *A* с 4
		003	264	B4		
		004	302	C2	JNZ	Если символ не равен 4, перейти к L1
		005	000	00	L1	
		006	003	03		
		007	333	DB	IN	Читать символ
		010	020	10		
		011	376	FE	CPI	Сравнить содержимое *A* с 5
		012	265	B5		
		013	302	C2	JNZ	Если символ не равен 5, перейти к L2
		014	002	02	L2	
		015	003	03		
		016	333	DB	IN	Читать символ
		017	020	10		
		020	376	FE	CPI	Сравнить содержимое *A* с 6
		021	266	B6		
		022	302	C2	JNZ	Если символ не равен 6, перейти к L2
		023	002	02	L2	
		024	003	03		
		025	016	OE	MVI C	Загрузить *n* в регистр *C*
		026	(*n*)	(*n*)	*n*	
		027	014	OC	INR C	Увеличить содержимое *C* (Установка флажков)
L4:		030	015	OD	DCR C	Уменьшить содержимое *C* (Установка флажков)
		031	312	CA	JZ	Если содержимое *C* равно 0, перейти к L3
		032	043	23	L3	
		033	003	03		
		034	333	DB	IN	Читать символ
		035	020	10		
		036	323	D3	OUT	Печатать символ
		037	030	18		
		040	303	C3	JMP	Перейти к L4
		041	030	18	L4	
		042	003	03		
L3:		043	166	76	HLT	Останов

манд (рис. 3.27 и 3.28) получаем восьмеричный и шестнадцатиричный листинги (табл. 3.5).

Решение для *Mc6800*.

Шаг 1
Шаг 2 } То же самое, что и для МП *In8080*.
Шаг 3

Шаг 4 — проектирование аппаратурной части. Структурная схема разрабатываемой системы на базе *Mc6800* показана на рис. 3.35. Как и в предыдущей задаче, в дополнение к порту вво-

да/вывода необходима схема *И* для формирования сигнала разрешения *е* (см. пример 15 в параграфе 3.4 схемы 1 и 2).

Шаг 5 — проектирование программного обеспечения. Непосредственно из схемы алгоритма (рис. 3.36) с использованием набора команд (рис. 3.30 и 3.31) получаем шестнадцатиричный листинг программы (табл. 3.6).

Таблица 3.6

Программа распечатки записи со считывателя по метке для *Мс 6800*

	16-й адрес		16-й код	Мнемокод	Комментарии
	H	L			
L1:	00	00	B6	LDA A, 2000	Читать символ
		01	20		
		02	00		
		03	01	NOP	
L2:		04	81	CMP A	Сравнить содержимое *A* с 4
		05	34		
		06	26	BNE L1	Если символ не равен 4, перейти к L1
		07	F8		
		08	B6	LDA A, 2000	Читать символ
		09	20		
		OA	00		
		OB	01	NOP	
		OC	81	CMP A	Сравнить содержимое *A* с 5
		OD	35		
		OE	26	BNE L2	Если символ не равен 5, перейти к L2
		OF	F4		
		10	B6	LDA A, 2000	Читать символ
		11	20		
		12	00		
		13	01	NOP	
		14	81	CMP A	Сравнить содержимое *A* с 6
		15	36		
		16	26	BNE L2	Если символ не равен 6, перейти к L2
		17	EC		
		18	C6	LDA B, *n*	Загрузить *n* в регистр *B*
		19	*n*	*n*	
L3:		1A	27	BEQ	Если содержимое *B* равно 0, перейти к L4
		1B	OB	L4	
		1C	B6	LDA A, 2000	Читать символ
		1D	20		
		1E	00		
		1F	01	NOP	
		20	B7	STA A, 4000	Печатать символ
		21	40		
		22	00		
		23	01	NOP	
		24	5A	DEC B	Уменьшить содержимое *B*
		25	7E	JMP L3	Перейти к L3
		26	00		
		27	1A		
L4:		28	3F	SWI	Останов

3.6. СПИСОК ЛИТЕРАТУРЫ

1. M6800 Microprocessor Applications Manual. Motorola, 1975.
2. Zissos D., Duncan F. G. Microprocessor Interfaces. — Electronic Letters, Vol. 12, № 23, November, 1976.
3. Zissos D. Problems and Solutions in Logis Design. Oxford University Press, 1976.
4. INTEL 8080 Microprocessor Systems User's Manual, September, 1975.
5. Zissos D., Bathory J. C. Wait/go Microprocessor Systems. — Proceedings Mimi 1977, November, 1977.

ГЛАВА 4

СИСТЕМЫ ПРОВЕРКИ И ПРОПУСКА

В этой главе рассмотрены алгоритмы проектирования и реализации микропроцессорных систем, построенных с применением режима проверки и пропуска. Обсуждается метод синхронизации ввода/вывода, основанный на растягивании синхроимпульса. Применяемая методика проектирования подробно изложена в параграфе 2.9, а собственно алгоритм проектирования — в 2.10.

4.1. БАЗОВЫЕ ОПРЕДЕЛЕНИЯ

Необходимость синхронизации МП с периферийными устройствами во время выполнения команд ввода/вывода была обоснована в гл. 2 и 3. Один из методов выполнения синхронизации ввода/вывода — метод ожидания/счета — был описан в предыдущей главе. В этой главе вниманию читателя предлагаются два альтернативных метода: проверки и пропуска и растягивания синхроимпульса.

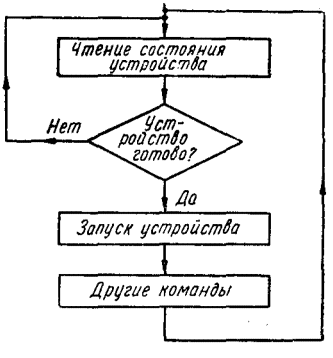


Рис. 4.1

В методе проверки и пропуска запрос МП синхронизируется с ответом периферийного устройства при помощи программного цикла. После выполнения каждой команды ввода/вывода программа опрашивает состояние внешнего устройства и анализирует его на окончание выполнения предыдущей операции. Если устройство не освободилось, оп-

рос повторяется. Этот программный цикл опроса повторяется до тех пор, пока устройство не станет готовым к обмену, что приведет к выходу из цикла и к выполнению следующей по порядку команды программы, как показано на рис. 4.1. Проектирование и реализация микропроцессорных систем с использованием этого метода описаны в следующем параграфе.

В редких случаях, когда синхронизация ввода/вывода может быть достигнута временным снижением частоты синхронизации, применяется метод, называемый растягиванием синхроимпульса. Он подробно изложен в параграфе 4.3.

4.2. СИСТЕМЫ ПРОВЕРКИ И ПРОПУСКА

Структурная схема системы проверки и пропуска показана на рис. 4.2. Сигнал *r* отображает доступность внешнего устройства. Алгоритм функционирования системы следующий. Микро-

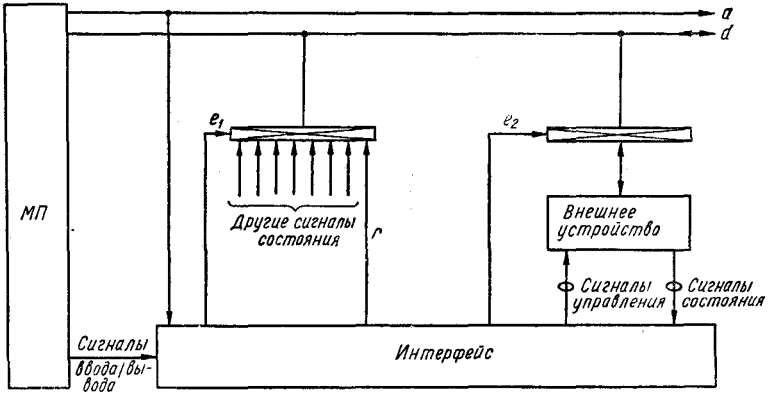


Рис. 4.2

процессор выполняет команду ввода/вывода, активизирующую периферийное устройство, в результате чего сигнал *r* изменяется от «1» до «0». Далее в МП вводится слово состояния, один из битов которого несет информацию о сигнале *r*. Затем этот бит проверяется на равенство «1» ($r=1$). Если *r* не равно 1, внешнее устройство еще не закончило выполнение отданного ему приказа, в МП опять вводится слово состояния и повторяется его проверка. Эта последовательность действий повторяется до тех пор, пока сигнал *r* не станет равным 1, после чего программа выходит из цикла (рис. 4.1). Отметим, что продолжительность работы в цикле проверки и пропуска равна времени ответа периферийного устройства.

Обозначим через f_{max} и f_{min} максимальную и минимальную частоты синхронизации, на которых может работать микропроцессор. Будем использовать переменную f для обозначения рабочей частоты, при этом

$$f_{min} \leqslant f \leqslant f_{max}.$$

Если время ответа внешнего устройства t, где

$$\frac{1}{f_{max}} \leqslant t \leqslant \frac{1}{f_{min}},$$

можно синхронизировать МП с внешним устройством, изменяя его частоту синхронизации до величины f, где

$$1/f \geqslant t.$$

Если предположить, что переходы микропроцессора из состояния в состояние происходят по спаду синхросигнала Φ_1, то целесообразно использовать синхросигнал Φ_2 для синхронизации в схеме растягивания синхроимпульса.

Реализация такой схемы весьма проста и не составляет трудности для читателя, обладающего практическим опытом логиче-

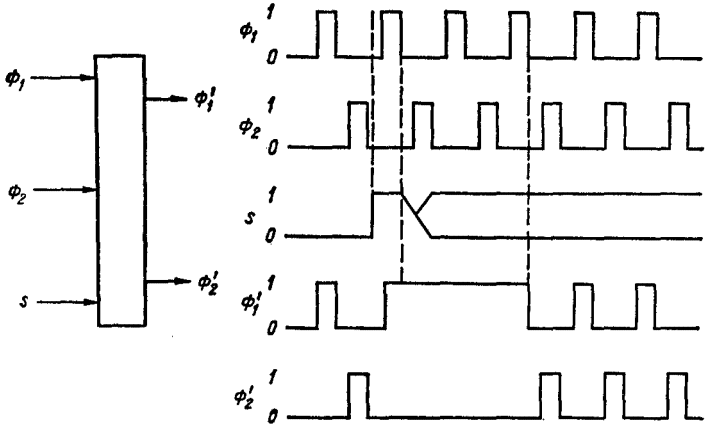


Рис. 4.3

ского проектирования. Сам метод проектирования описан ранее в гл. 1.

Для иллюстрации метода покажем, как можно растянуть синхроимпульсы микропроцессора на два синхропериода первоначальной частоты. Если обозначить через S сигнал, генерируемый интерфейсом для запроса двухциклового растягивания, то тогда вид рассматриваемых сигналов будет соответствовать изображенному на рис. 4.3. Диаграмма состояния схемы показана на

рис. 4.4. Непосредственно из нее получаем:

$$S_A = S_1 = \bar{A}B, \text{ поэтому } J_A = B;$$

$$R_A = S_3\bar{s} = A\bar{B}\bar{s}, \text{ поэтому } K_A = \bar{B}\bar{s};$$

$$S_B = S_0 s = \bar{A}\bar{B}s, \text{ поэтому } J_B = \bar{A}s;$$

$$R_B = S_2 = AB, \text{ поэтому } K_B = A;$$

$$\Phi_1' = S_0\Phi_1 + S_1 + S_2 + S_3\Phi_1 = \bar{A}\bar{B}\Phi_1 + \bar{A}B + AB + A\bar{B}\Phi_1 = B + \Phi_1;$$

$$\Phi_2' = (S_0 + S_3)\,\Phi_2 = (\bar{A}\bar{B} + A\bar{B})\,\Phi_2 = \bar{B}\Phi_2.$$

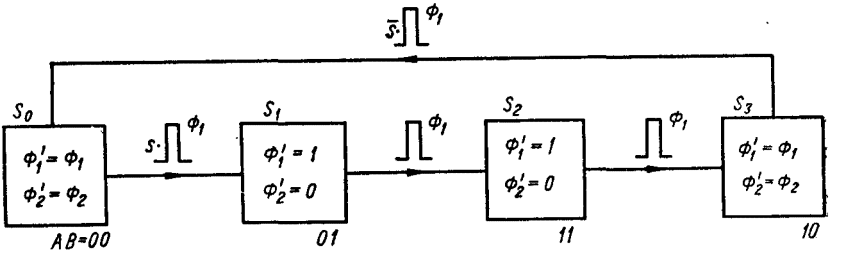


Рис. 4.4

Соответствующая схема показана на рис. 4.5.

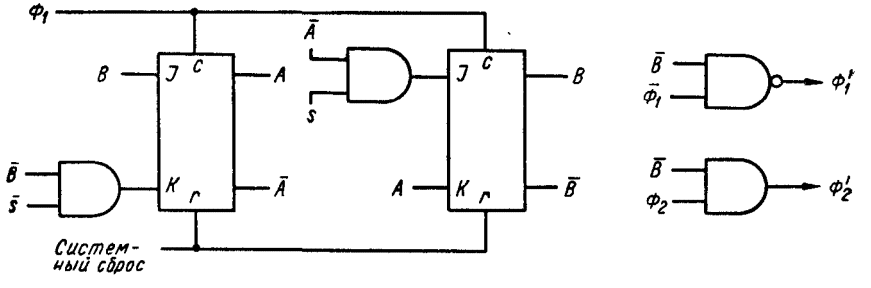


Рис. 4.5

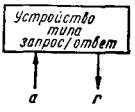
4.4. ЗАДАЧИ И РЕШЕНИЯ

В данном разделе рассматриваются задачи и их решения для алгоритма построения систем проверки и пропуска. Обратим внимание на тот факт, что приведенные решения для микропроцессора *In 8080* не ограничивают область применения алгоритма для других типов микропроцессоров. Первые три шага алгоритма проектирования можно выполнить вообще безотносительно к типу МП.

Задача 1. Распечатка содержимого ОЗУ. Спроектировать интерфейс, позволяющий программно пересылать данные из последовательных ячеек ОЗУ или ПЗУ через МП к приемнику инфор-

мации. Приемником информации является устройство цифровой печати, технические характеристики которого указаны на рис. 4.6. Используется МП *In 8080.*

Устройство типа запрос/ответ



Контакт *a*. Переход «0» → «1» на этом контакте активизирует устройство (подает запрос).

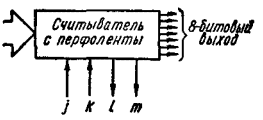
Контакт *r*. Сигнал состояния *r* равен «1», когда устройство готово к обмену, и равен «0» в противоположном случае. Активизация устройства недопустима, если $r = 0$.

Печатающее устройство

Контакт *w*. Импульс отрицательной полярности длительностью свыше 1 мкс загружает символ в буфер либо запускает механизм печати, если буфер заполнен, или вводится символ «возврата каретки» (015 в коде *ASC II* (см. рис. 3.23)). Недопустимые в коде *ASC II* символы игнорируются.

Контакт *x*. Сигнал состояния *x* равен 1, когда печатающее устройство готово, и на шине *w* уровень 1.

Считыватель



Контакт *j*. Подача «0» на этот контакт запускает прогон ленты влево.

Контакт *k*. Подача «0» на этот контакт вызывает прогон ленты вправо.

Контакт *l*. Сигнал состояния *l* устанавливается в 1, когда отверстие синхродорожки перфоленты находится под считывающей головкой, иначе $l = 0$.

Контакт *m*. Уровень «1» на этом контакте указывает, что считыватель готов к приему следующего импульса прогона. Для остановки перфоленты на очередном символе необходимо снять импульс прогона на 1 мс сразу за фронтом сигнала *l*. Минимальная длительность импульса *l* 1 мкс.

Рис. 4.6

Решение.

Шаг 1 — формирование технического задания. Целью проектирования является построение интерфейса между МП и относительно простым периферийным устройством в режиме проверки и пропуска для используемого микропроцессора.

Шаг 2 — внешнее проектирование. Сигналы ввода/вывода МП *In 8080,* используемые в рассматриваемом случае, показаны на рис. 4.7, технические характеристики печатающего устройства — на рис. 4.6.

Шаг 3 — системное проектирование. Структурная схема разрабатываемой системы показана на рис. 4.8, а ее функционирование описывается алгоритмом, показанным на рис. 4.9.

Шаг 4 — разработка аппаратурной части. Необходимые аппаратурные средства содержат: адресный дешифратор, элемент *И—НЕ,* инвертор и элемент *И* (рис. 4.10). Дешифратор используется для декодирования адреса ввода/вывода печатающего устройства и трехстабильного буфера. Можно использовать один

и тот же адрес, так как буфер работает в режиме ввода, а печатающее устройство — в режиме вывода. Если обозначить адрес через A_m, то $w=\overline{A_m \mathrm{OUT}\,\mathrm{WR}}$; этот сигнал вырабатывается

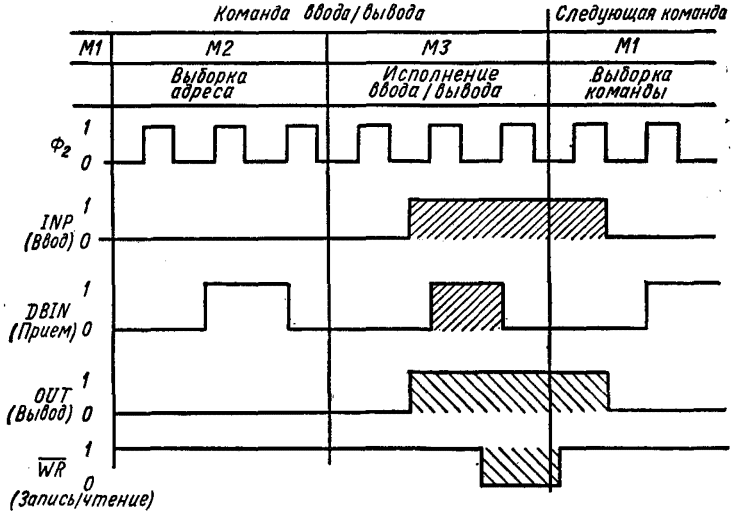


Рис. 4.7

схемой *И—НЕ*. Трехстабильный буфер открывается при выполнении команды *IN* (ввод) с адресом A_m, т. е. $e=\mathrm{IN}\,\mathrm{DBIN}\,A_m$. Этот сигнал формируется вентилем *И*. Так как сигнал состояния

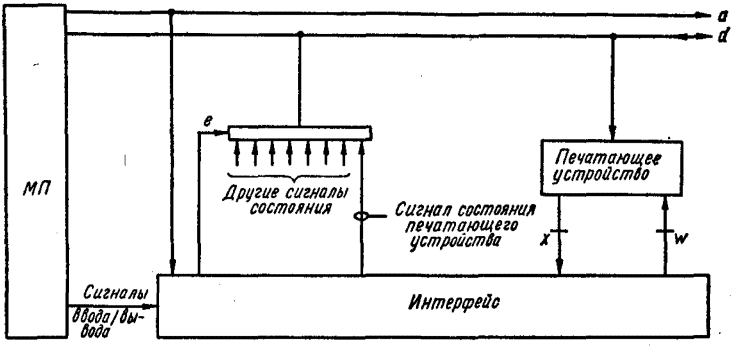


Рис. 4.8

печатающего устройства x может передаваться только при $w=1$, то можно в качестве сигнала w использовать сигнал $\overline{\mathrm{OUT}}$. Это сократит интервал времени, в течение которого сигнал w не может передаваться, до величины длительности импульса OUT.

Как ранее было оговорено, все 8 выходов трехстабильного буфера хранят состояние «0», пока печатающее устройство не готово.

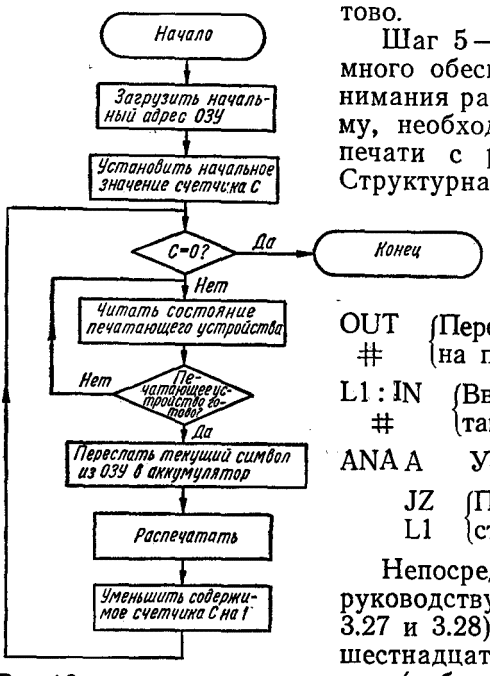


Рис. 4.9

Шаг 5 — проектирование программного обеспечения. Для простоты понимания рассмотрим сначала программу, необходимую для синхронизации печати с работой микропроцессора. Структурная схема этой программы показана на рис. 4.1. Сама программа на языке ассемблера написана ниже.

OUT #	Передать символ из АС на печатающее устройство
L1 : IN #	Ввести состояние с печатающего устройства
ANA A	Установить флажки
JZ L1	При неготовности устройства вернуться к метке L1.

Непосредственно из схемы рис. 4.9, руководствуясь системой команд (рис. 3.27 и 3.28), запишем восьмеричный и шестнадцатиричный листинги программы (табл. 4.1) для адреса внешнего устройства A_m=A010.

Задача 2. Считать информацию и ввести в ОЗУ. Спроектировать интерфейс для программной пересылки данных от внешне-

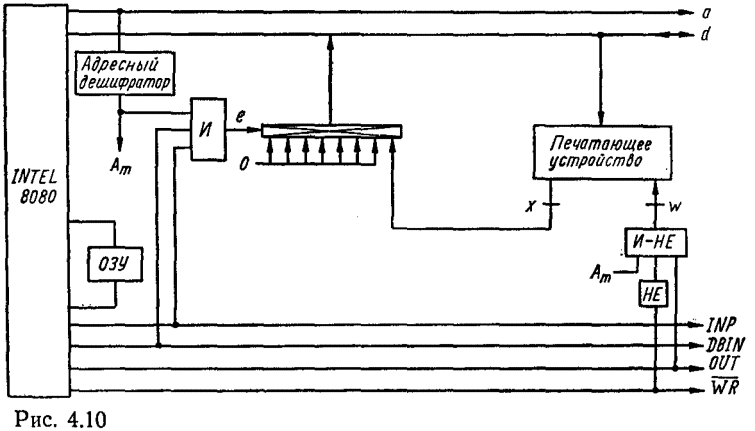


Рис. 4.10

го источника через микропроцессор в последовательно расположенные ячейки ОЗУ.

Таблица 4.1

Программа распечатки содержимого ОЗУ

	8-й адрес		8-й код	16-й код	Мнемокод	Комментарии
	H	L				
	003	000	041	21	LXI H	Загрузить непосредственно начальный адрес ОЗУ в регистровую пару HL
		001	(L)	(L)		
		002	(H)	(H)		
		003	016	OE	MVI C	Загрузить длину блока в регистр *C*
		004	(*n*)	(*n*)		
		005	014	OC	INR C	Увеличить содержимое регистра *C* для установки флажков
L3:		006	015	OD	DCR C	Уменьшить содержимое регистра *C*
		007	312	CA	JZ	Если содержимое регистра *C*=0, перейти к L1
		010	027	17	L1	
		011	003	03		
L2:		012	333	DB	IN	Читать состояние принтера
		013	010	08		
		014	247	A7	ANA A	Установить флажки
		015	312	CA	JZ	Если принтер не готов, перейти к L2
		016	012	OA	L2	
		017	003	03		
		020	176	7E	MOV A,M	Переслать следующий символ из памяти в аккумулятор
		021	323	03	OUT	Переслать символ на принтер
		022	010	08		
		023	054	2C	INR L	Увеличить адрес ОЗУ
		024	303	C3	JMP	Перейти к L3
		025	006	06	L3	
		026	003	03		
L1:		027	076	3E	MVI A	Загрузить в аккумулятор непосредственно код 015_8 для возврата каретки
		030	015	8D		
		031	323	D3	OUT	Печать. Содержимое аккумулятора распечатывается, и каретка возвращается
		032	010	08		
		033	166	76	HLT	Останов

Источником данных является считыватель с перфоленты, описанный на рис. 4.6; в качестве микропроцессора используется *In 8080.*

Решение.

Шаг 1 — формирование технического задания. Задачей проектирования является построение интерфейса между микропроцессором и внешним устройством, для которого существует ограничение на временные соотношения входных сигналов.

Шаг 2 — внешнее проектирование. Используемые сигналы ввода/вывода микропроцессора *In 8080* показаны на рис. 4.7, структурная схема фотосчитывателя и назначение его выводов — на рис. 4.6.

Шаг 3 — системное проектирование. Структурная схема разрабатываемой системы показана на рис. 4.11, а схема алгоритма

ее функционирования — на рис. 4.12. Все 8 выводов трехстабильного буфера состояния находятся в состоянии 0, пока считыватель не готов к обмену.

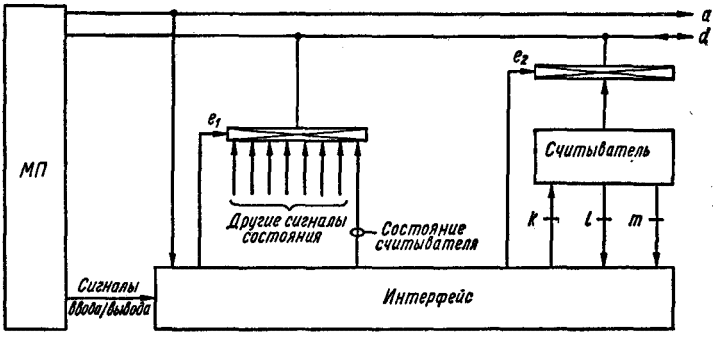


Рис. 4.11

Шаг 4 — проектирование аппаратурной части. В связи с тем, что минимальная длительность импульса ввода/вывода микропроцессора составляет 0,48 мкс, его нельзя непосредственно передать на фотосчитыватель. Это препятствие можно преодолеть на практике с помощью триггера, устанавливаемого импульсом ввода/вывода.

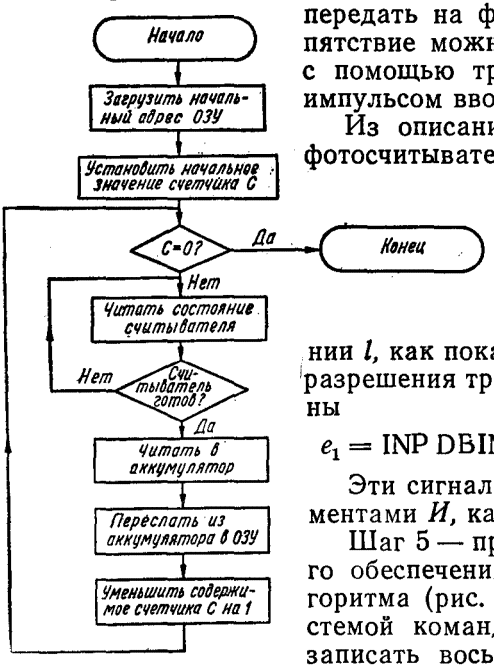


Рис. 4.12

Из описания характеристик выводов фотосчитывателя следует, что сигнал прогона ленты может быть отключен, когда сигнал l сбрасывается в «0». Таким образом, триггер можно сбрасывать переходом из «1» в «0» сигнала на линии l, как показано на рис. 4.13. Сигналы разрешения трехстабильных буферов равны

$$e_1 = \text{INP DBIN } A_q; \qquad e_2 = \text{INP DBIN } A_p.$$

Эти сигналы формируются двумя элементами *И*, как показано на рис. 4.13.

Шаг 5 — проектирование программного обеспечения. Непосредственно из алгоритма (рис. 4.14) и в соответствии с системой команд (рис. 3.27, 3.28) можно записать восьмеричный и шестнадцатиричный листинги программы для адресов $A_p = A_{020}$, $A_q = A_{03b}$ (табл. 4.2).

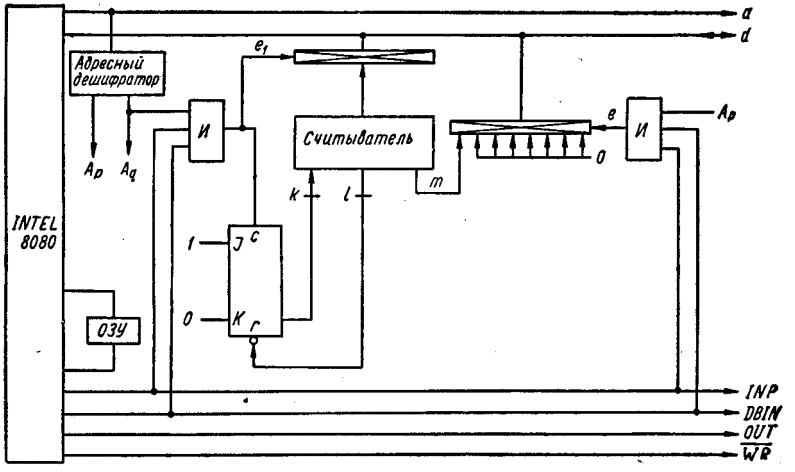


Рис. 4.13

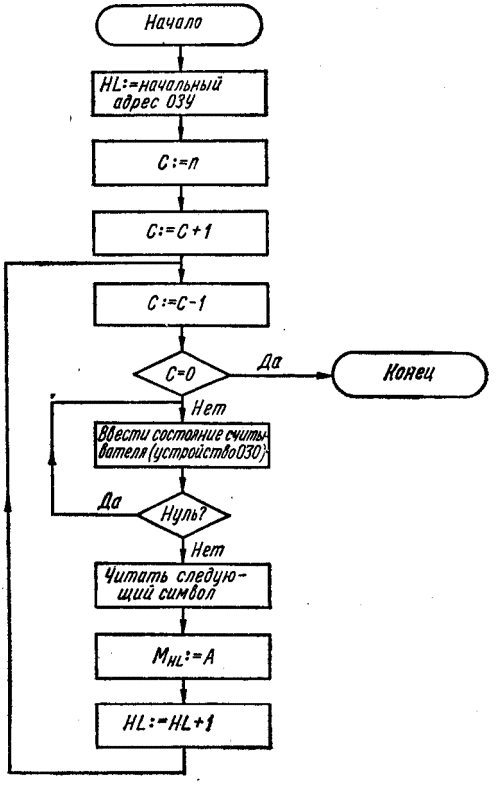


Рис. 4.14

Таблица 4.2

Программа ввода данных в ОЗУ с фотосчитывателя

	8-й адрес		8-й код	16-й код	Мнемокод	Комментарии
	H	L				
	003	000	041	21	LXI H	Загрузить начальный адрес ОЗУ в регистровую пару HL
		001	(L)	(L)		
		002	(H)	(H)		
		003	016	OE	MVI C	Загрузить длину блока в регистр *C*
		004	(*n*)	(*n*)		
		005	014	OC	INR C	Увеличить содержимое регистра C для установки флажков
L3:		006	015	OD	DCR C	Уменьшить содержимое регистра *C*
		007	312	CA	JZ	Если содержимое регистра *C*=0, перейти к L1
		010	027	17	L1	
		011	003	03		
L2:		012	333	DB	IN	Читать состояние считывателя
		013	020	10		
		014	247	A7	ANA A	Установить флажки
		015	312	CA	JZ	Если считыватель не готов, перейти к L2
		016	012	OA	L2	
		017	003	03		
		020	333	DB	IN	Читать следующий символ
		021	030	18		
		022	167	77	MOV M,A	Загрузить содержимое аккумулятора в память
		023	054	2C	INX H	Увеличить адрес памяти
		024	303	C3	JMP	Перейти к L3
		025	006	06	L3	
		026	003	03		
L1:		027	166	76	HLT	Останов

Задача 3. Прочитать и отпечатать *n* символов. Спроектировать интерфейс между считывателем с перфоленты, описанным в задаче 2, и печатающим устройством, описанным в задаче 1, для распечатки программно-устанавливаемого числа считываемых символов. Использовать микропроцессор *In 8080*.

Решение.

Шаг 1 — формирование технического задания. Цель проектирования — построение интерфейса между МП и двумя внешними устройствами.

Шаг 2 — внешнее проектирование. Необходимая информация о сигналах ввода/вывода микропроцессора *In 8080* приведена на рис. 4.7, а характеристики сигналов сопряжения фотосчитывателя и печатающего устройства — на рис. 4.6.

Шаг 3 — системное проектирование. Структурная схема разрабатываемой системы показана на рис. 4.15. Используемое здесь решение включает: чтение символа, распечатку символа, уменьшение счетчика на 1 и ожидание готовности считывателя и печатающего устройства для повторения процедуры. После то-

го как содержимое счетчика приобретает значение 0, указывая тем самым, что n символов переданы и отпечатаны, работа заканчивается, как показано на схеме алгоритма (рис. 4.16).

Шаг 4 — проектирование аппаратурной части. Руководствуясь описанием сигналов сопряжения фотосчитывателя, находим, что

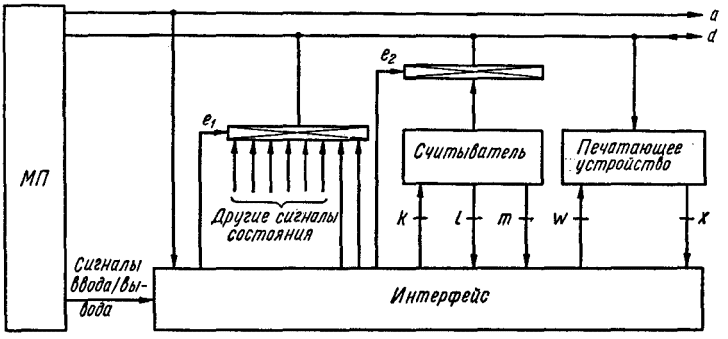


Рис. 4.15

для продвижки перфоленты на один символ необходимо подать нулевой сигнал на контакт k длительностью не менее 1 мкс, и снять его перед тем, как сигнал l станет равным «1». Так как длительность импульса ввода/вывода примерно равна 0,5 мкс, его нельзя непосредственно использовать для работы с фотосчитывателем. Наиболее простым решением в данном случае является применение триггера, устанавливаемого импульсом ввода/вывода и сбрасываемого сигналом l при переходе из «1» в «0». Этот переход происходит через 2 мс после подачи «0» на контакт k. Таким образом, длительность сигнала k примерно равна 2 мс. Присвоим считывателю адрес ввода/вывода 003 и будем использовать сигнал INP DBINA_{303} для перевода JK-триггера из «1» в «0». Этот же сигнал используется для стробирования трехстабильного буфера фотосчитывателя, т. е.

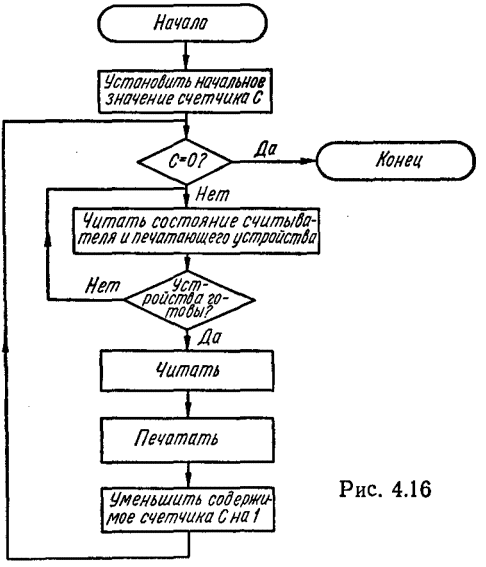


Рис. 4.16

$$e_1 = \text{INP DBIN } A_{003}.$$

Для запуска печатающего устройства нулевой сигнал подается на контакт w при выполнении команды ввода/вывода с адресом 004, т. е. $w = \overline{\text{OUT WR}A_{004}}$. Этот сигнал вырабатывается элементом *И—НЕ* и инвертором, показанными на рис. 4.17. Сигналы

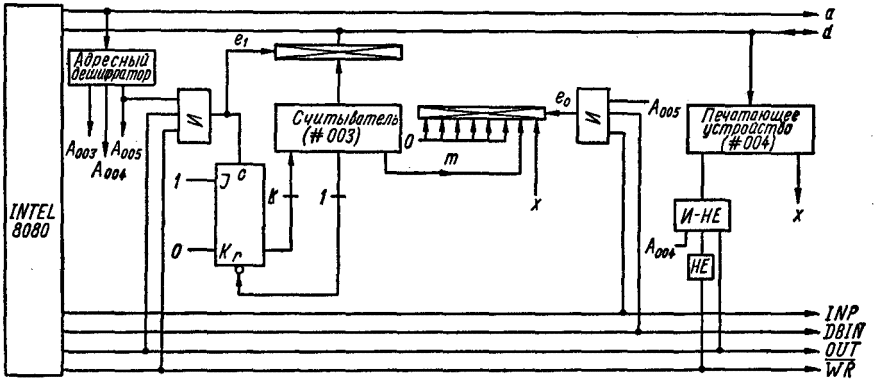


Рис. 4.17

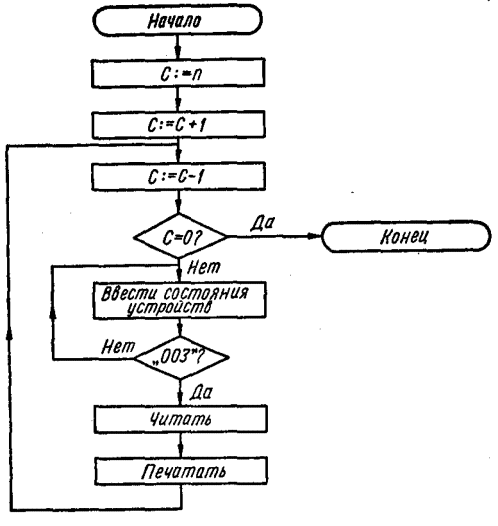


Рис. 4.18

состояния m и x подаются на второй трехстабильный буфер. Выходы этого буфера несут информацию о готовности считывателя и печатающего устройства, причем если устройства не готовы, то на выходах буфера формируется код 000 000 11 или 003_8. Это дает возможность программисту проверять готовность обоих устройств.

Для нашего случая порту состояния присвоен адрес 005.

Шаг 5 — проектирование программного обеспечения. По схеме алгоритма (рис. 4.18) и в соответствии с таблицей команд *In 8080* (рис. 3.27 и 3.28) строим восьмеричный и шестнадцатиричный листинги программы (табл. 4. 3).

Таблица 4.3

Программа распечатки *n* символов с фотосчитывателя

	8-й адрес		8-й код	16-й код	Мнемокод	Комментарии
	H	L				
	003	000	016	01	MVI	Загрузить следующий байт в регистр *C*, *n* — число символов
		001	(*n*)	(*n*)		
		002	014	0C	INR C	Увеличить содержимое регистра *C* для установки флажков условий
L3:		003	015	0D	DCR C	Уменьшить содержимое регистра *C*
		004	312	CA	JZ	Если содержимое регистра *C*=0, перейти к L1
		005	025	15	L1	
		006	003	03		
L2:		007	333	DB	IN	Ввести состояние
		010	005	05		
		011	376	FE	CPI	Сравнить содержимое аккумулятора со следующим байтом
		012	003	03		
		013	302	C2	JNZ	Если устройство не готово, перейти к L2
		014	007	07	L2	
		015	003	03		
		016	333	DB	IN	Читать символ
		017	003	03		
		020	323	D3	OUT	Печатать символ
		021	004	04		
		022	303	C3	JMP	Перейти к L3
		023	003	03	L3	
		024	003	03		
L1:		025	166	76	HLT	Останов

ГЛАВА 5

СИСТЕМЫ ПРЕРЫВАНИЙ

В этой главе разработаны алгоритмы проектирования и реализации систем, работающих в режиме прерывания. Методика проектирования изложена в гл. 2 (2.9 и 2.10).

5.1. БАЗОВЫЕ ОПРЕДЕЛЕНИЯ

Способность системы к обработке прерываний означает, что некоторые внешние события могут вызвать временное прекращение выполнения фоновой программы микропроцессором с целью

отработки некоторой другой последовательности команд, именуемой программой обработки прерываний (рис. 5.1). После того как запрос прерывания удовлетворен, возобновляется обработка прерванной фоновой программы. Например, датчик пожара может сообщить микропроцессору, что обнаружено загорание; в таком случае микропроцессор должен прервать выполнение текущей программы и выполнить служебную программу (например, включением сигнала тревоги предостеречь работающий поблизости персонал, включить систему пожаротушения, вызвать пожарную команду и т. д.). Ответив таким образом на пожарный сигнал, микропроцессор может вернуться к обработке прерванной программы.

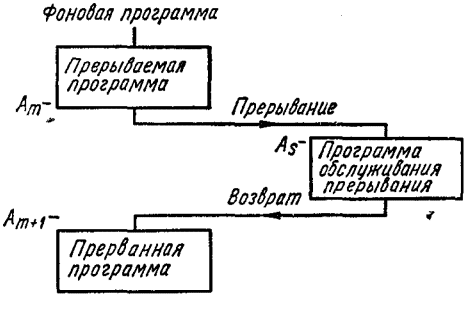


Рис. 5.1

Введем следующие обозначения: A_m — ячейка памяти, содержащая последнюю выполненную команду фоновой программы к моменту прерывания; A_s — ячейка, содержащая первую команду программы обработки прерываний. В момент прерывания содержимое программного счетчика *PC* равно A_{m+1}, так как оно увеличивается в машинном цикле *M1* (см. рис. 2.4) выполняемой команды.

Для того чтобы переключиться с основной программы на подпрограмму обработки прерываний, необходимо заменить содержимое программного счетчика *PC* на A_s. Аналогично для возврата в прерванную программу по окончании обработки прерываний необходимо загрузить программный счетчик *PC* величиной A_{m+1}. Последняя представляет собой минимальную информацию, необходимую микропроцессору для возврата в прерванную программу. Будем называть эту информацию адресом возврата в программу. На практике необходимо сохранять на время обработки прерываний также содержимое рабочих регистров и флажков состояния. Под рабочими регистрами понимаются регистры, используемые как прерванной программой, так и программой обработки прерывания. Содержимое программного счетчика, флажков состояния и рабочих регистров в дальнейшем будем называть информацией возврата в программу [1].

Примерами применения режима прерывания программы на практике могут служить запуск, сопровождение и завершение некоторого процесса, протекающего параллельно (одновременно) с выполнением основной программы.

Хотя разработка систем прерывания также выполняется по обобщенному алгоритму проектирования, реализация таких си-

стем требует существенно большего объема аппаратуры и программного обеспечения, чем любой другой режим работы микропроцессора.

5.2. СИСТЕМЫ ПРЕРЫВАНИЙ

Структурная схема системы обработки прерываний показана на рис. 5.2. Она состоит из логики прерываний (схема обработки прерываний), используемой всеми устройствами, работающими в режиме прерывания; периферийных устройств и интерфейса

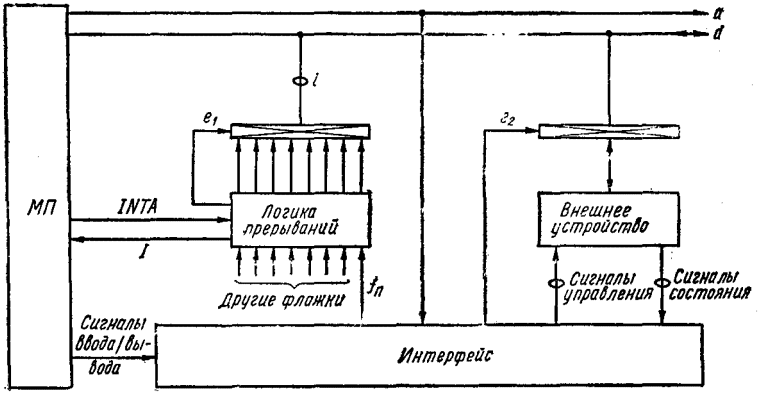


Рис. 5.2

прерываний. Приведенная схема представляет собой обобщенный вариант системы обработки прерываний без конкретизации типа микропроцессора и внешних устройств. Схема алгоритма ее работы изображена на рис. 5.3, а описание его приведено ниже.

Когда периферийное устройство требует обслуживания или готово передавать данные в микропроцессор (или принять их из МП), его интерфейс, осуществляющий связь с системой, устанавливает флажок (концепция флажков будет описана далее). Схема обработки прерываний, воспринимая состояние всех флажков, генерирует сигнал прерывания (сигнал *I* на рис. 5.2) и некоторую дополнительную информацию, которую обозначим переменной *i*. Сигнал *I* сообщает микропроцессору, что одно или несколько внешних устройств желают установить связь с микропроцессором. Если прерывания разрешены, микропроцессор заканчивает текущую команду и отвечает на запрос следующим образом.

1. Микропроцессор генерирует сигнал INTA (рис. 5.2), указывающий, что выполняемая программа прервана. Будем называть его сигналом подтверждения прерывания.

2. Дальнейшие запросы прерывания не воспринимаются (запрещаются). Таким образом, процесс выполнения программы

обработки прерывания микропроцессором не будет прерван до тех пор, пока это снова не станет возможным по логике программы.

3. Информация возврата запоминается в стеке.

4. Определяется источник прерывания путем ввода и расшифровки информации *i*.

5. Содержимое рабочих регистров запоминается в стеке.

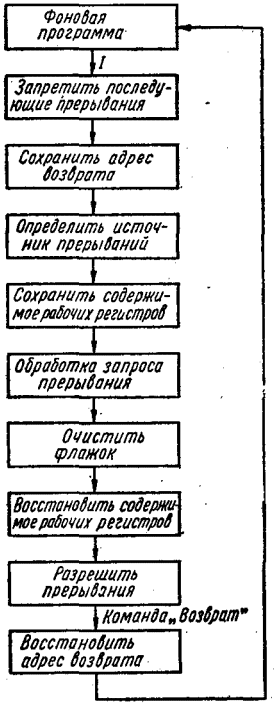


Рис. 5.3

6. Обслуживается запрос прерывания.

7. Сбрасывается флажок запроса прерывания.

8. Восстанавливается содержимое рабочих регистров.

9. Разрешаются прерывания.

10. Осуществляется выход из подпрограммы обработки прерывания.

(Возврат в прерванную программу путем загрузки в *PC* адреса возврата).

Приведенная выше последовательность событий является обобщенной и может в определенной степени варьироваться в зависимости от конкретного микропроцессора, на котором реализуется система. Например, для микропроцессора *Mc6800* все рабочие регистры микропроцессора и флажки состояния заносятся в стек автоматически, тогда как в микропроцессоре *In 8080* автоматически в стеке запоминается только содержимое программного счетчика. В данном случае задача запоминания содержимого рабочих регистров и флажков условий должна решаться пользователем.

Конфигурация систем обработки прерывания для различных микропроцессоров различна и обусловлена структурой цикла обработки прерывания. Как будет показано в 5.4, разработка системы прерывания не представляет существенных трудностей, если разработчик знаком со структурой цикла обработки прерываний конкретного микропроцессора. Однако, прежде чем переходить к детализированному описанию процесса проектирования систем прерывания, представляется целесообразным изложить концепцию флажков и принципа их сортировки, что будет сделано в следующем параграфе.

5.3. ФЛАЖКИ И ИХ СОРТИРОВКА

Флажки. Определим флажок как сигнал, генерируемый и используемый некоторым устройством для информирования другого устройства о запросе на установление связи. Структурная схема цепи, способной устанавливать, сбрасывать, разрешать и запрещать выдачу флажков, показана на рис. 5.4. Функции четырех входных сигналов следующие. Входной сигнал e разрешает доступ к флажку, а сигнал d запрещает его. Эти два сигнала не должны появляться одновременно. Когда доступ к флажку разрешен, сигнал на входе k устанавливает его. Флажок сбрасывается (очищается) сигналом на входе c. Если нет необходимости в управлении, доступном к флажку, входы d и e могут не использоваться.

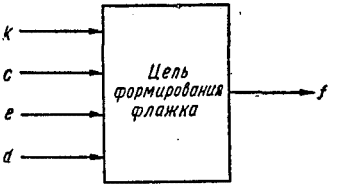


Рис. 5.4

Схема флажка. Как и для всякой логической схемы, передаточная характеристика логической структуры флажка может быть реализована с помощью различных (но функционально

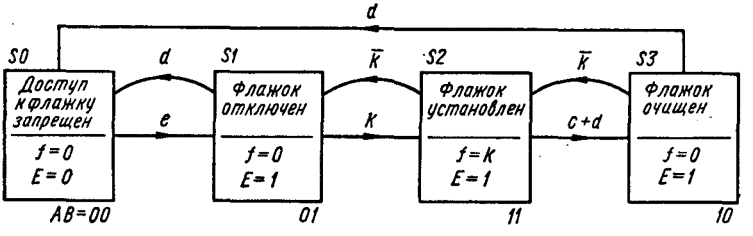


Рис. 5.5

эквивалентных) схем. Ниже мы рассмотрим два возможных способа реализации флажков, именуемых в дальнейшем флажок 1 и флажок 2.

Флажок 1. Диаграмма состояний, описывающая внутренние и внешние операции схемы [2], изображена на рис. 5.5. Непосредственно из этой диаграммы получаем:

условие установки $A = Bk$;

условие сброса $A = B\overline{k} + \overline{B}d \xrightarrow{\text{инверсия } A} (\overline{B} + k)(B + \overline{d})$;

условие установки $B = \overline{A}e + A\overline{k}$;

условие сброса $B = \overline{A}d + Ac + Ad = d + Ac \xrightarrow{\text{инверсия } B} \overline{d}(\overline{A} + \overline{c})$.

Таким образом,

$$A = Bk + A(\overline{B} + k)(B + \overline{d});$$
$$B = \overline{A}e + A\overline{k} + B(\overline{A} + \overline{c})\overline{d};$$
$$f = S_2k = ABk.$$

Эквивалентная схема *И—НЕ* показана на рис. 5.6.

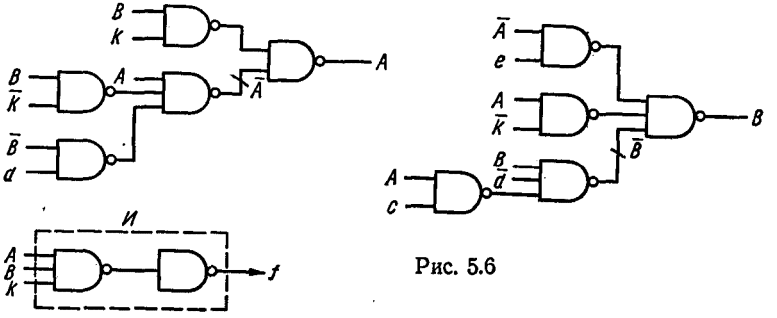


Рис. 5.6

Флажок 2. Если отсутствует необходимость в цепях разрешения/запрета, а сигнал k является импульсным, флажок можно реализовать в виде триггера, как показано на рис. 5.7 (см. также 1.10).

Идентификация флажков. Как отмечалось выше, сигнал прерывания формируется путем выполнения функции *ИЛИ* над со-

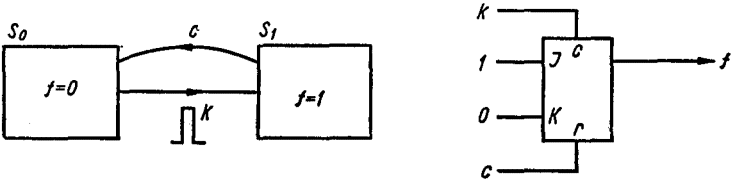


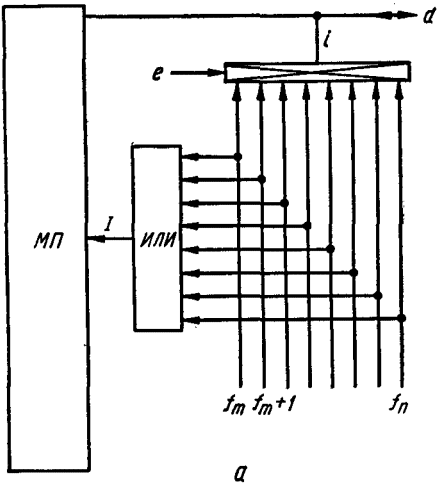
Рис. 5.7

держимым всех флажков. Этот сигнал просто информирует микропроцессор о запросе прерывания от одного или нескольких устройств. Таким образом, когда микропроцессор начинает обрабатывать прерывание, подпрограмма обработки должна идентифицировать источник прерывания. Существует два основных метода идентификации флажков: последовательный и векторный. Каждый из этих методов рассмотрим подробнее.

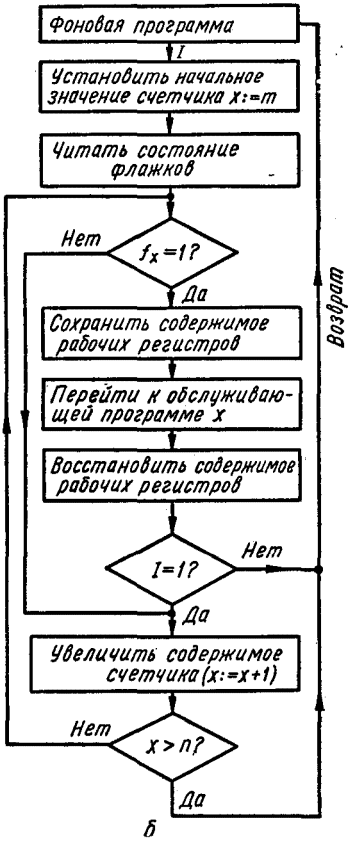
Последовательный метод. В этом случае после получения запроса прерывания микропроцессор последовательно просматривает устройства, пока не обнаружит устройство, требующее обслуживания. Когда такое устройство найдено, просмотр прекращается и вызывается соответствующая подпрограмма обслужи-

вания. Если после окончания обслуживания среди непросмотренных устройств еще есть устройства, выставившие запрос, просмотр продолжается, в противном случае производится возврат в прерванную программу.

Существует несколько способов практической реализации последовательного метода. Для описываемого далее примера флажки подсоединяются ко входному порту, через который их содержимое читается микропроцессором во внутренний регистр по шине данных (рис. 5. 8, *а*). Содержимое внутреннего регистра затем проверяется побитно, как показано на рис. 5.8,*б*.



а



б

Рис. 5.8

Векторный метод. В этом методе с помощью специальной логической схемы — сортировщика флажков (приоритетного шифратора) автоматически устанавливается наличие флажков, идентифицируются выставившие их устройства (см. рис. 5.9, *а*). Сигнал прерывания *I* генерируется, как и во всякой системе, путем выполнения операции *ИЛИ* над содержимым флажков. Идентифицируется флажок специальным двоичным кодом. Будем использовать в наших примерах прямой двоичный код (8—4—2—1), если не оговорено особо.

Проектирование и реализация приоритетных шифраторов не представляют трудностей для читателей, знакомых с материалом,

изложенным в гл. 1. Продемонстрируем алгоритм проектирования на примерах двух-, восьми- и шестидесятичетырехфлажковых шифраторов. При этом будем полагать, что более высоким

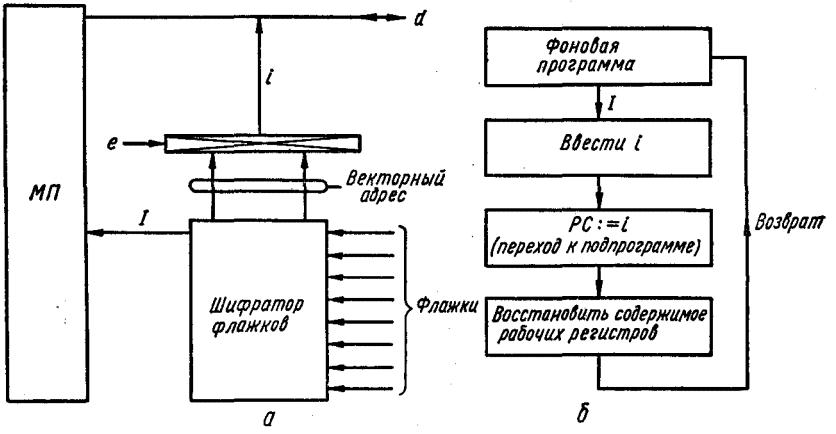


Рис. 5.9

приоритетом обладает устройство с большим флажковым номером.

Двухфлажковый приоритетный шифратор. Структурная схема двухфлажкового шифратора показана на

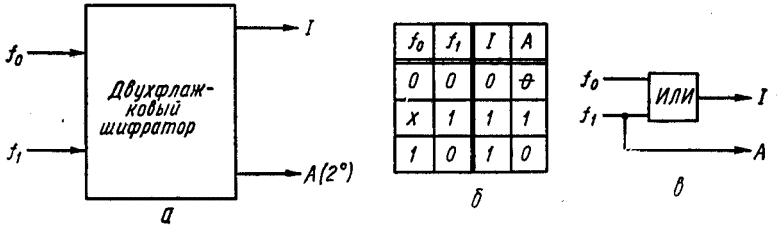


Рис. 5.10

рис. 5.10, *а*, а его передаточная функция (характеристика вход/выход) — на рис. 5.10, *б* в виде таблицы истинности. Непосредственно из этой таблицы получаем следующие уравнения:

$$I = f_0 + f_1; \quad A = f_1.$$

Схема, реализующая эти функции, показана на рис. 5.10, *в*.

Восьмифлажковый шифратор. Структурная схема восьмифлажкового шифратора показана на рис. 5.11, *а*, характеристика вход/выход в виде таблицы истинности — на рис. 5.11, *б*. Непосредственно из этой таблицы получаем следующие

уравнения:

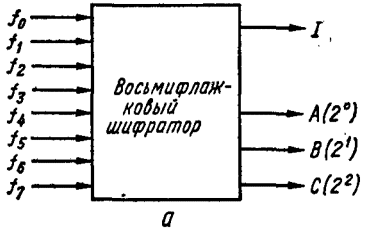
$$I = f_0 + f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7;$$

$$A = f_7 + \bar{f}_7\bar{f}_6 f_5 + \bar{f}_7\bar{f}_6\bar{f}_5\bar{f}_4 f_3 + \bar{f}_7\bar{f}_6\bar{f}_5\bar{f}_4\bar{f}_3\bar{f}_2 f_1 =$$
$$= f_7 + \bar{f}_6 f_5 + \bar{f}_6\bar{f}_4 f_3 + \bar{f}_6\bar{f}_4\bar{f}_2 f_1;$$

$$B = f_7 + \bar{f}_7 f_6 + \bar{f}_7\bar{f}_6\bar{f}_5\bar{f}_4 f_3 + \bar{f}_7\bar{f}_6\bar{f}_5\bar{f}_4\bar{f}_3 f_2 =$$
$$= f_7 + f_6 + \bar{f}_5\bar{f}_4 f_3 + \bar{f}_5\bar{f}_4 f_2;$$

$$C = f_7 + \bar{f}_7 f_6 + \bar{f}_7\bar{f}_6 f_5 + \bar{f}_7\bar{f}_6\bar{f}_5 f_4 = f_7 + f_6 + f_5 + f_4.$$

В связи с тем, что восьмивходовые приоритетные шифраторы выпускаются в виде БИС, их реализация на вентилях не показана.



а

Входы								Выходы			
f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	I	C	B	A
0	0	0	0	0	0	0	0	0	0	0	0
x	x	x	x	x	x	x	1	1	1	1	1
x	x	x	x	x	x	1	0	1	1	1	0
x	x	x	x	x	1	0	0	1	1	0	1
x	x	x	x	1	0	0	0	1	1	0	0
x	x	x	1	0	0	0	0	1	0	1	1
x	x	1	0	0	0	0	0	1	0	1	0
x	1	0	0	0	0	0	0	1	0	0	1
1	0	0	0	0	0	0	0	1	0	0	0

б

Рис. 5.11

Шестидесятичетырехфлажковый шифратор. Структурная схема 64-флажкового шифратора показана на рис. 5.12. Флажки объединены в 8 групп по 8 флажков в каждой, причем каждая группа связана со своим шифратором. Выходные сигналы восьми приоритетных шифраторов поданы на вход селектора групп, являющегося также приоритетным шифратором. Работает схема следующим образом.

Селектор группы выбирает группу, в которой есть установленный флажок, генерирует сигнал прерывания для системы *I* и трехбитовый адрес, идентифицирующий выбранную группу. Сигналы *D*, *E* и *F*, кроме того, что они подсоединены к адресной шине, подаются также на вход двоично-десятичного дешифратора. Каждый из восьми выходов дешифратора стробирует три трехстабильных буфера, соединяющих адресную шину с соответствующим приоритетным шифратором, как показано на рис. 5.12. Заметим, что данный 64-флажковый шифратор можно использовать для обслуживания меньшего числа флажков путем простого заземления неиспользованных входов.

Очевидно, что модульный метод, используемый при реализации 64-флажкового шифратора на основе 8-флажковых шифра-

торов, можно применять для разработки системы, обслуживающей до 4096 флажков путем установки 64-флажкового шифратора в качестве модуля в схеме рис. 5.12.

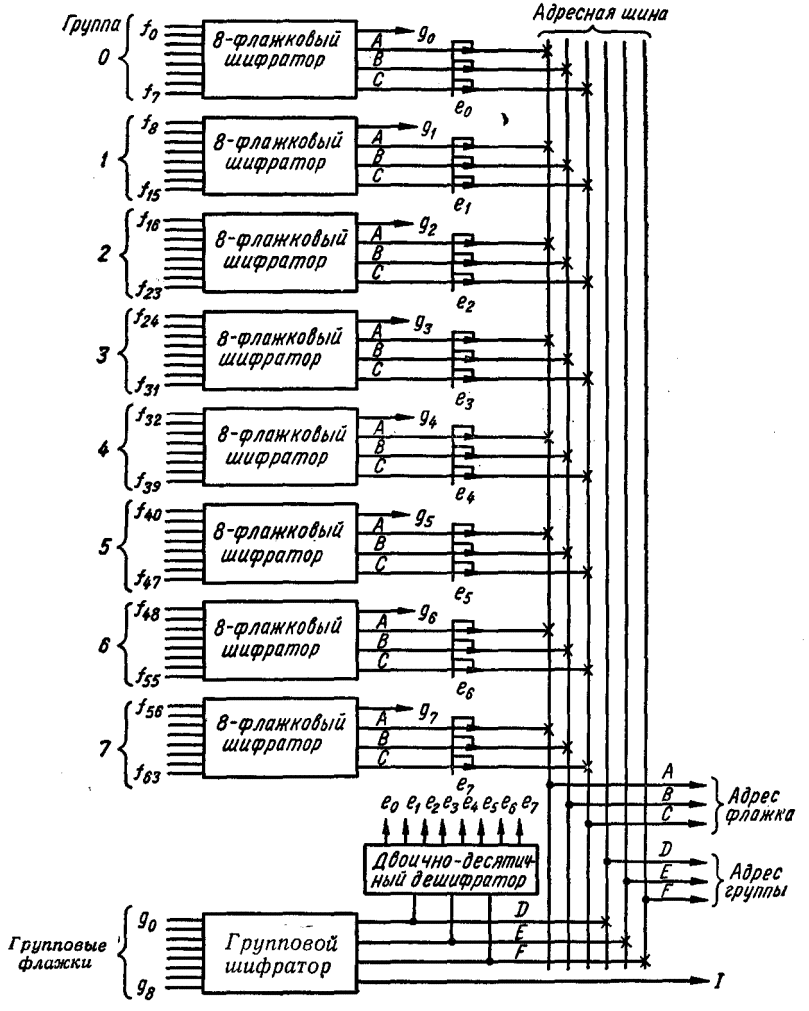


Рис. 5.12

5.4. СИСТЕМА ПРЕРЫВАНИЙ МИКРОПРОЦЕССОРА *In 8080*

В этом параграфе проектируется система прерываний для микропроцессора *In 8080*. Как было указано выше, для проектирования системы прерываний необходимо ознакомиться со струк-

турой цикла обработки прерывания в микропроцессоре, который, как мы увидим, в значительной мере индивидуален. Рассмотрим этот вопрос более детально.

Цикл прерывания в микропроцессоре *In 8080*[3]. Функциональное назначение выводов МП, приведенное на рис. 2.9, показывает, что подача логической «1» на контакт *14* прерывает программу, если прерывания не запрещены. Разрешение прерывания индицируется логической «1» на контакте *16*. На входной сигнал запроса прерывания не налагаются никакие временные ограни-

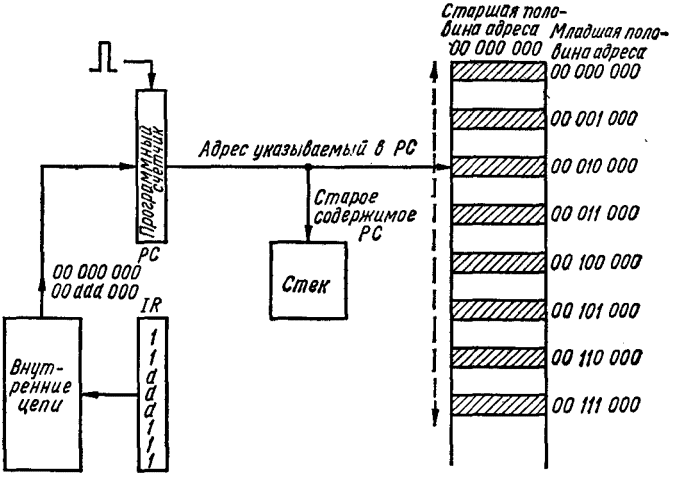


Рис. 5.13

чения — сигнал можно подавать в любое время. Это возможно благодаря наличию внутренней синхронизации запроса прерывания в микропроцессоре путем его стробирования синхроимпульсом Φ_2 на последнем такте командного цикла, в течение которого поступил запрос прерывания.

Как уже отмечалось, метод, используемый в МП *In 8080*, весьма своеобразен. Этот метод проще понять, будучи знакомым с методом отработки микропроцессором команды *RST*. Код операции этой команды 11*ddd*111. Тремя буквами *ddd* обозначено восьмеричное число от 0 до 7. Как и для команд ввода/вывода, на отработку команды *RST* затрачивается три машинных цикла.

Предположим, что команда *RST* загружена в регистр кода операции *IR* (рис. 5.13). В течение следующих трех машинных циклов содержимое программного счетчика *PC* засылается в стек (по 8 бит за цикл). Параллельно со стековыми действиями содержимое регистра команд (11*ddd*111) подается на вход внутренних цепей, генерирующих два 8-битовых слова:

00 000 000 и 00 *ddd* 000.

По мере того как байты программного счетчика засылаются в стек, эти сгенерированные слова занимают их место, причем 00 000 000 заносится в старший байт программного счетчика *PC*, а 00*ddd*000 — в младший байт. Таким образом, следующая команда будет извлечена из одной из следующих 8 ячеек:

00 000 000 (000_8) 00 000 000 (000_8), если *ddd* = 000;
00 000 000 (000_8) 00 001 000 (010_8), если *ddd* = 001;
00 000 000 (000_8) 00 010 000 (020_8), если *ddd* = 010;
00 000 000 (000_8) 00 011 000 (030_8), если *ddd* = 011;
00 000 000 (000_8) 00 100 000 (040_8), если *ddd* = 100;
00 000 000 (000_8) 00 101 000 (050_8), если *ddd* = 101;
00 000 000 (000_8) 00 110 000 (060_8), если *ddd* = 110;
00 000 000 (000_8) 00 111 000 (070_8), если *ddd* = 111.

Как ясно из предыдущего параграфа, трехразрядная двоичная переменная *ddd* в команде *RST* может быть сгенерирована приоритетным шифратором.

Ниже будет показано, как команда *RST* (повторного запуска) может использоваться для прерывания микропроцессора

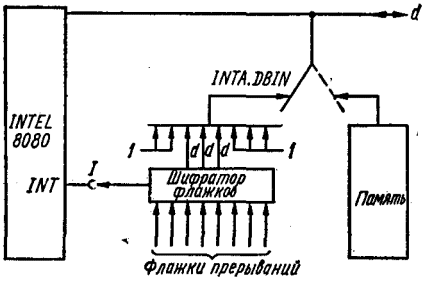


Рис. 5.14

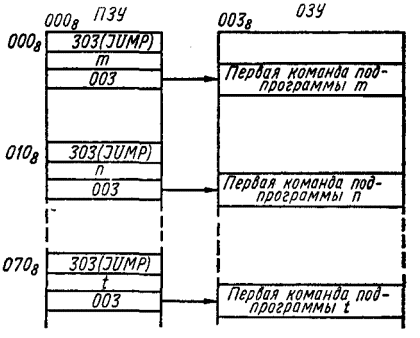


Рис. 5.15

In 8080 в аварийных ситуациях, даже когда прерывания запрещены. Эта особенность, аналогичная немаскируемому прерыванию в МП *Mc6800*, является полезной в случаях, связанных с высоким риском, таких как управление и контроль в медицине и др.

Возвратимся теперь к циклу прерывания МП *In 8080*. Он похож на обычный цикл извлечения команды (см. рис. 2.2, 2.4), за исключением того что: а) генерируется сигнал подтверждения прерывания; б) содержимое программного счетчика не увеличивается на 1; в) запрещаются дальнейшие прерывания.

Если в течение времени действия сигналов INTA DBIN отключить память от шины данных и перейти к обработке команды

RST, как показано на рис. 5.14, управление программой передается одной из 8 ячеек, показанных на рис. 5.13, в зависимости

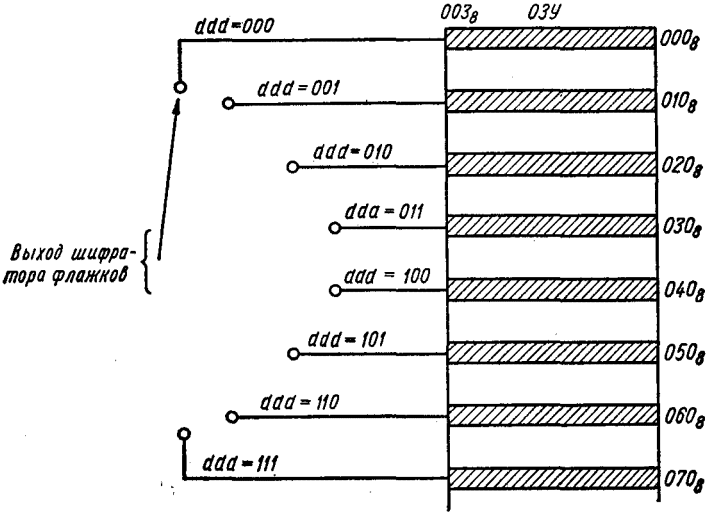


Рис. 5.16

от кода *ddd*, выдаваемого приоритетным шифратором. Например, если на выходе приоритетного шифратора будет код 010, то управление передается ячейке с адресом 00 010 00 (020).

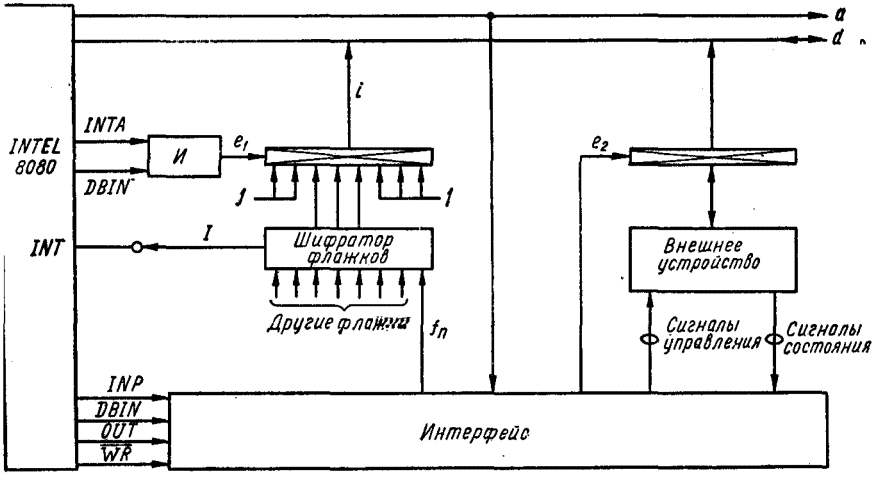


Рис. 5.17

В существующих системах обычно (хотя и не всегда) ячейки ПЗУ имеют адреса с нулевым старшим байтом. Так как подпро-

граммы обслуживания прерывания, написанные пользователем, должны располагаться в ОЗУ (из-за невозможности изменения содержимого ПЗУ), то ячейки ПЗУ с адресами 00 *ddd* 000 должны содержать команды перехода в соответствующие ячейки ОЗУ (см. рис. 5.15). Для рассматриваемого примера будем полагать, что ячейки ОЗУ начинаются с адресов, у которых старший байт равен 00 000 011 (003), а ячейки ОЗУ, в которые передается управление при переходе, имеют младший байт адреса от 000_8 до 070_8. С точки зрения пользователя это означает, что сигнал прерывания вызывает передачу управления на одну из восьми ячеек вектора ОЗУ в зависимости от величины кода *ddd* (рис. 5.16). Любая из восьми ячеек может использоваться для первой команды программы обслуживания прерывания. Если программа обслуживания имеет более восьми команд, то программист может использовать команду *JUMP* для перехода на другие адреса памяти.

Содержимое программного счетчика на момент поступления прерывания восстанавливается командой возврата. Содержимое рабочих регистров восстанавливается путем выполнения соответствующих команд извлечения из стека. Структурная схема микропроцессорной системы прерываний на базе *In 8080* в окончательном виде изображена на рис. 5.17.

5.5. ЧРЕЗВЫЧАЙНЫЕ ПРЕРЫВАНИЯ МИКРОПРОЦЕССОРА *In 8080*

Одной из характерных особенностей любой системы, работающей в медицинском, промышленном и другом оборудовании, эксплуатируемом в условиях высокого риска, является ее способ-

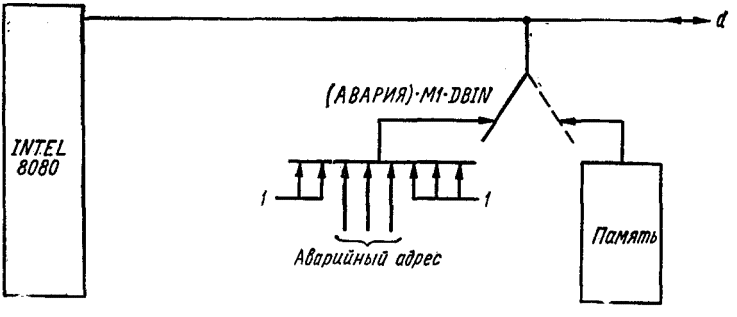


Рис. 5.18

ность отвечать на запросы в критических ситуациях с минимальной задержкой. В МП *In8080,* как указывалось выше, при входе в цикл обработки прерывания автоматически запрещаются все остальные прерывания. Для разрешения их необходимо применение команды «разрешение прерывания» *EI.* До тех пор, пока

такая команда не выполнена, микропроцессор будет игнорировать внешние запросы прерывания, т. е. аварийные запросы не будут рассмотрены в течение этого времени. Этот интервал времени может затянуться до опасных пределов из-за того, что программист не учел необходимости обработки аварийных сигналов либо забыл поставить в необходимом месте команду разрешения прерывания.

Риск появления такой ситуации можно уменьшить с помощью выдачи на шину команд команды *RST* при появлении аварийного запроса с соответствующей величиной *ddd* (рис. 5.18). Это вызовет практически мгновенный переход на подпрограмму обработки аварийного запроса независимо от того, запрещены прерывания или нет.

5.6. СИСТЕМА ПРЕРЫВАНИЙ МИКРОПРОЦЕССОРА *Mc 6800*

Как будет видно ниже, для микропроцессора *Mc6800* можно построить две системы прерывания. Однако вначале рассмотрим структуру цикла прерывания этого микропроцессора.

Цикл прерывания *Mc 6800.* Согласно схеме функционального назначения выводов (см. рис. 2.9), работа микропроцессора

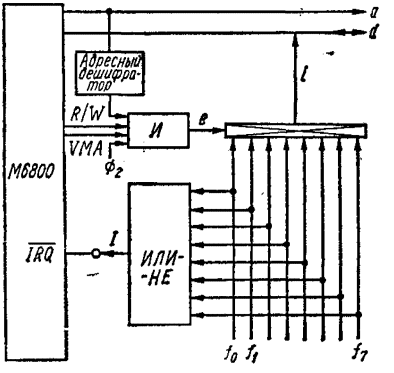


Рис. 5.19

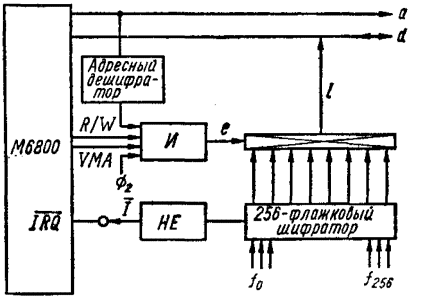


Рис. 5.20

Mc6800 может быть прервана подачей логического «0» на контакт *14,* если прерывания не запрещены. Как и для *In8080,* на сигналы запроса прерывания не налагается никаких временных ограничений, т. е. они могут появиться в любой момент. Это возможно благодаря тому, что синхронизация с обрабатываемой командой достигается внутренним стробированием запроса прерывания фазой Φ_2 в последнем такте командного цикла.

Процесс обслуживания прерывания происходит следующим образом:

1) заканчивается текущая выполняемая команда;
2) запрещаются дальнейшие прерывания;
3) состояние микропроцессора автоматически записывается в стек в следующем порядке: PC_L, $PC_н$, IR_L, $IR_н$, АССА, АССВ, СС;
4) программный счетчик загружается величиной, записанной в ячейках FFF8 ($PC_н$) и FFF9 (PC_L). Эти ячейки, как уже объяснялось, содержат адрес первой команды программы обработки прерываний.

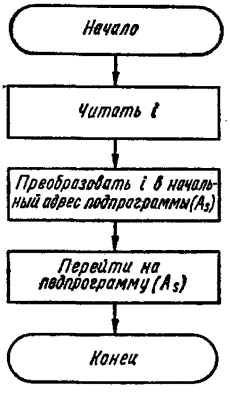


Рис. 5.21

Внимание читателя обращается на тот факт, что на данном этапе источник прерывания не идентифицируется.

Возврат в прерванную программу в конце программы обработки прерывания выполняется с помощью команды возврат из прерывания (RTI), код операции которой $3B_{16}$. В результате выполнения данной команды восстанавливается состояние микропроцессора до прерывания путем извлечения из стека содержимого регистров микропроцессора в порядке, обратном занесению.

Источник прерывания *Mc6800* можно определить как последовательным, так и векторным методом, описанными в 5.3. Рассмотрим две разновидности системы обработки прерываний.

Система прерываний 1. В этой системе используется последовательный метод для идентификации источника прерывания. Его структурная схема показана на рис. 5.19, а алгоритм его функционирования описан на рис. 5.8, б.

Система прерываний 2. В системе 2 используется векторный метод для идентификации источника прерывания. Структурная схема системы показана на рис. 5.20, а алгоритм функционирования — на рис. 5.21.

5.7. ЭКСТРЕННЫЕ ПРЕРЫВАНИЯ МИКРОПРОЦЕССОРА *Mc 6800*

Экстренные прерывания обслуживаются после подачи сигнала на вход *NMI (немаскируемое прерывание)* (контакт *39*, рис. 2.9).

5.8. ЗАДАЧИ И РЕШЕНИЯ

В данном параграфе продемонстрируются применения общего алгоритма проектирования при решении задач вплоть до уровня окончательных решений. Внимание читателя обращается на тот факт, что приводимые процедуры проектирования пригодны для любого микропроцессора, хотя в примерах использовались

только *In8080* и *Mc6800*. Отдельно хочется отметить, что первые три шага алгоритма проектирования выполняются без учета типа применяемого микропроцессора.

Задача 1. Счетчик событий. Импульсы, представляющие собой некоторые события, возникают случайным образом на шине *q* (рис. 5.22). Спроектировать систему прерывания, позволяющую распечатать содержимое счетчика событий всякий раз при ручном нажатии кнопки *m*. Нажатие на кнопку сбрасывает счетчик (предполагается, что нажатие происходит нечасто).

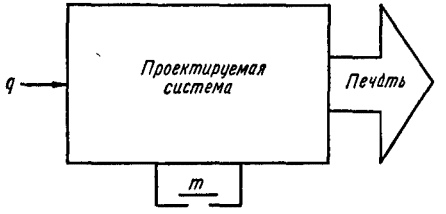


Рис. 5.22

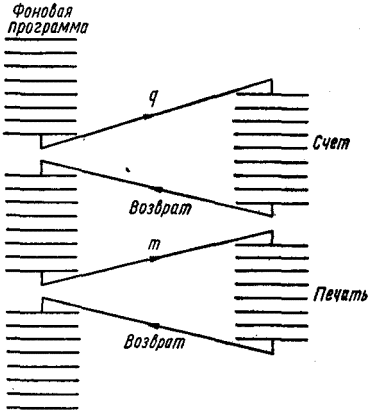


Рис. 5.23

Систему реализовать на микропроцессоре *In8080* (задача 1) и микропроцессоре *Mc6800* (задача 2).

Доступные разработчику адреса ввода/вывода следующие:

для *In8080* — 003_8 и 004_8;

для *Mc6800* — 2000_{16}, 8004_{16} и 8006_{16}.

Решение для *In8080*.

Шаг 1 — формирование технического задания. Цель проектирования состоит в выдаче содержимого счетчика событий по запросам с использованием режима прерывания, как показано на рис. 5.23.

Шаг 2 — внешнее проектирование. Структурная схема системы прерывания на микропроцессоре *In8080* показана на рис. 5.17, а сигналы ввода/вывода — на рис. 4.7.

Характеристики сопряжения печатающего устройства приведены на рис. 5.24.

Шаг 3 — системное проектирование. Структурная схема общего решения показана на рис. 5.25, а ее функционирование — на схеме алгоритма рис. 5.26. Каждый импульс на шине *q* прерывает текущую программу и увеличивает содержимое счетчика на 1. Аналогично при нажатии клавиши прерывается текущая программа и выполняется другая служебная подпрограмма, обеспечивающая печать содержимого счетчика и сброс его содержимого в ноль.

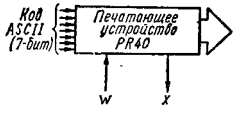
Шаг 4 — проектирование аппаратурной части. Из структурной схемы рис. 5.27 (общей структурной схемы разрабатываемого

Устройство типа запрос/ответ

Контакт *a*. Переход «0» → «1» на этом контакте активизирует устройство (подает запрос).

Контакт *r*. Сигнал состояния *r* равен «1», когда устройство готово к обмену, и равен «0» в противоположном случае. Активизация устройства недопустима, если $r = 0$.

Печатающее устройство



Контакт *w*. Импульс отрицательной полярности длительностью свыше 1 мкс загружает символ в буфер либо запускает механизм печати, если буфер заполнен, или вводится символ «возврата каретки» (015 в коде *ASCII* (см. рис. 3. 23)). Недопустимые в коде *ASCII* символы игнорируются.

Контакт *x*. Сигнал состояния *x* равен 1, когда печатающее устройство готово, и на шине *w* уровень 1.

Считыватель

Контакт *j*. Подача «0» на этот контакт запускает прогон ленты влево.

Контакт *k*. Подача «0» на этот контакт вызывает прогон ленты вправо.

Контакт *l*. Сигнал состояния *l* устанавливается в 1, когда отверстие синхродорожки перфоленты находится под считывающей головкой, иначе $l = 0$.

Контакт *m*. Уровень «1» на этом контакте указывает, что считыватель готов к приему следующего импульса прогона. Для остановки перфоленты на очередном символе необходимо снять импульс прогона на 1 мс сразу за фронтом сигнала *l*. Минимальная длительность импульса *l* 1 мкс.

Рис. 5.24

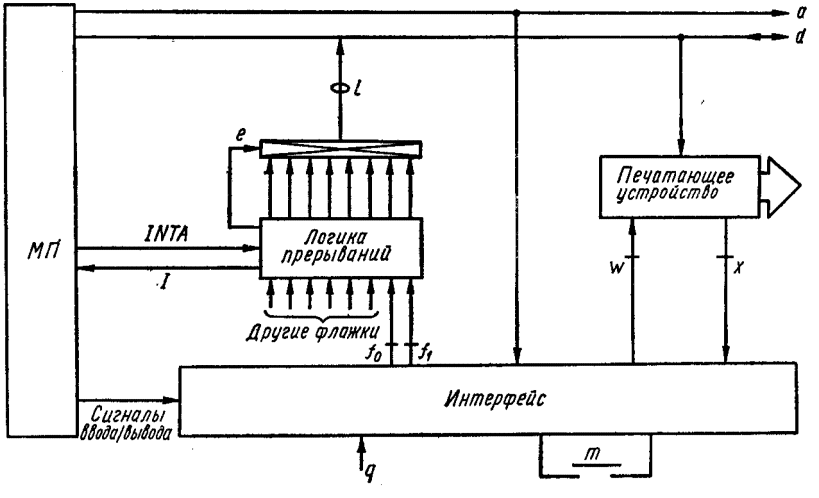


Рис. 5.25

устройства на МП *In 8080*) видно, что интерфейс генерирует следующие сигналы:

1) два флажка — f_0 и f_1;
2) импульс печати w.

Если нет необходимости разрешать или запрещать доступ к флажкам, то они могут формироваться схемой типа 2 (см. 5.2).

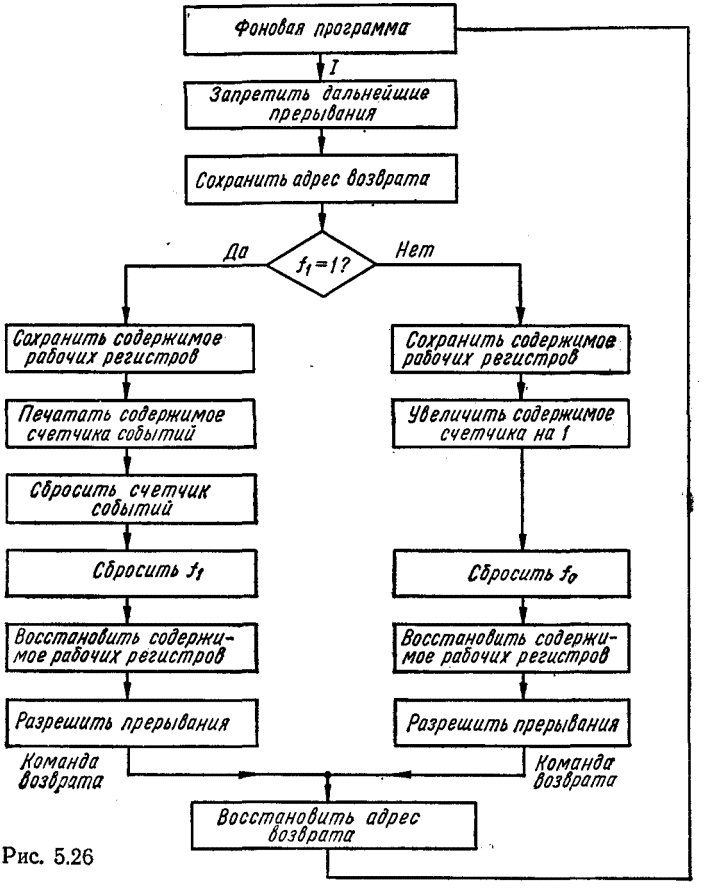


Рис. 5.26

Если использовать импульс q для установки триггера f_0 (рис. 5.28), а сигнал с кнопки подать на триггер f_1, то уравнения интерфейсных сигналов примут вид:

$$c_0 = q;$$
$$r_0 = \overline{\text{OUT WR } A_{003}};$$
$$c_1 = m;$$
$$r_1 = w;$$
$$w = \overline{\text{OUT WR } A_{004}}.$$

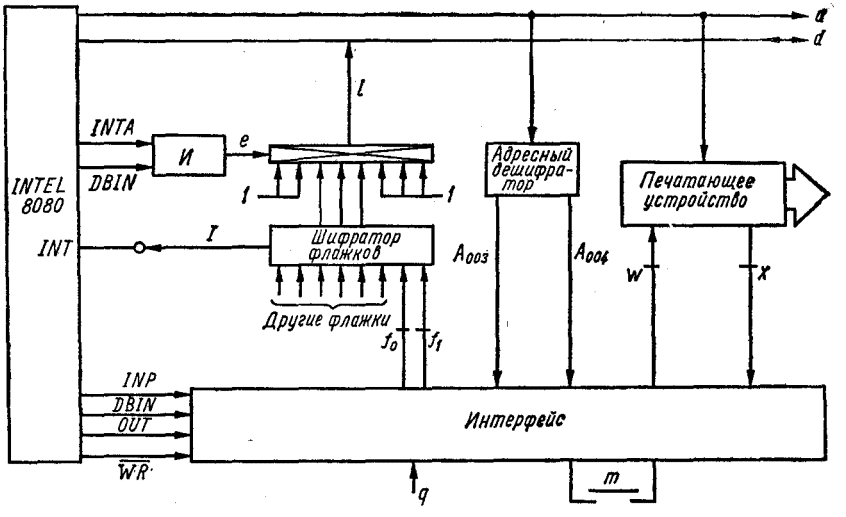


Рис. 5.27

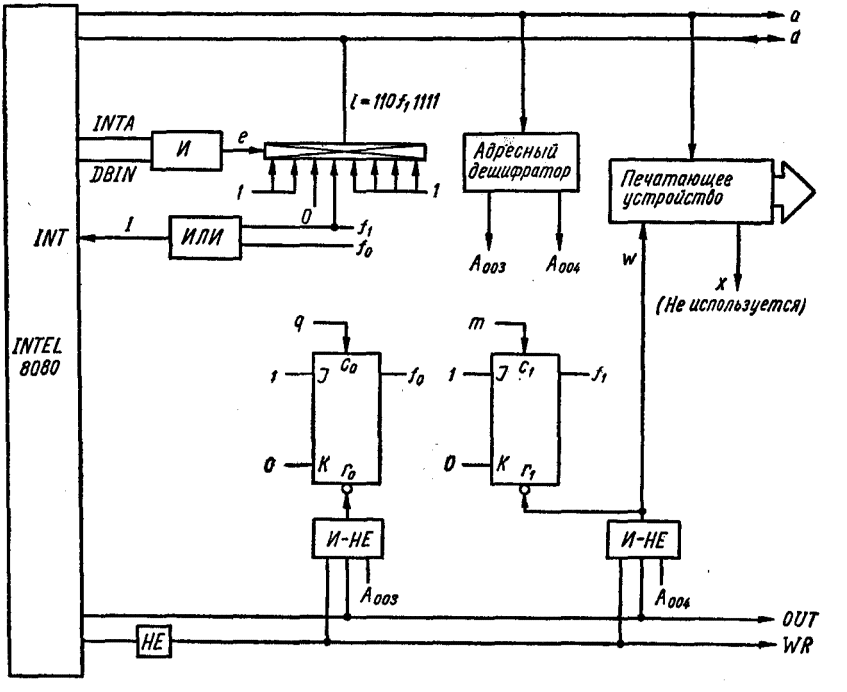


Рис. 5.28

Далее, если принять во внимание, что f_0 и f_1 — единственные флажки системы, то можно удовлетвориться двухфлажковым приоритетным шифратором. Как показано в 5.3, такая схема реализуется на вентиле *ИЛИ* (см. рис. 5.10). При этих условиях аппаратурная часть системы имеет конфигурацию, показанную на рис. 5.28.

Шаг 5 — проектирование программного обеспечения. Структурная схема алгоритма для *In 8080* показана на рис. 5.29 (см. также рис. 5.3). Непосредственно из алгоритмов, пользуясь приведенной на рис. 3.28 системой команд МП, получаем следующие восьмеричный и шестнадцатиричный листинги программы (табл. 5.1).

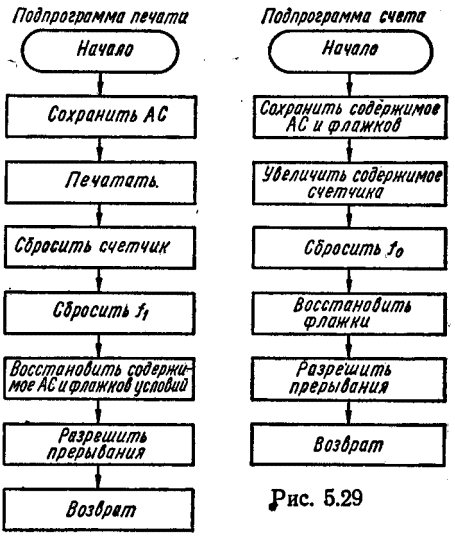


Рис. 5.29

Таблица 5.1

Подпрограмма обработки прерываний в счетчике событий на *In 8080*

8-й адрес		8-й код	16-й код	Мнемокод	Комментарии
				Подпрограмма обслуживания 0 (Счет)	
H	L				
003	010	365	F5	PUSH PSW	Сохранить содержимое *A* и флажков
	011	004	04	INR B	Увеличить содержимое *B*
	012	323	D3	OUT	Цикл ввода/вывода для очистки флажка f_0
	013	003	03		
	014	361	F1	POP PSW	Восстановить *A* и флажки
	015	373	FB	EI	Разрешить прерывания
	016	311	C9	RET	Возврат
				Подпрограмма обслуживания 1 (Печать)	
	030	365	F5	PUSH PSW	Сохранить содержимое *A* и флажков
	031	170	78	MOV A, B	Переслать из *B* в *A*
	032	323	D3	OUT	Цикл вывода для загрузки принтера и очистки флажка f_1
	033	004	04		
	034	076	3E	MVI A	Загрузить в *A* код возврата каретки в *ASCII*
	035	015	0D	015	
	036	323	D3	OUT	Опрос принтера
	037	004	04		
	040	006	06	MVI B	Очистка регистра *B* счетчика
	041	060	30		
	042	361	F1	POP PSW	Восстановить *A* и флажки
	043	373	FB	EI	Разрешить прерывания
	044	311	C9	RET	Возврат

Решение для *Mc6800.*

Шаг 1
Шаг 2 } То же самое, что и для МП *In8080.*
Шаг 3

Шаг 4 — проектирование аппаратурной части.

Для МП *Mc6800* разработчик имеет возможность выбора между реализацией системы обработки прерываний с последовательным или векторным методами идентификации. В приведенном

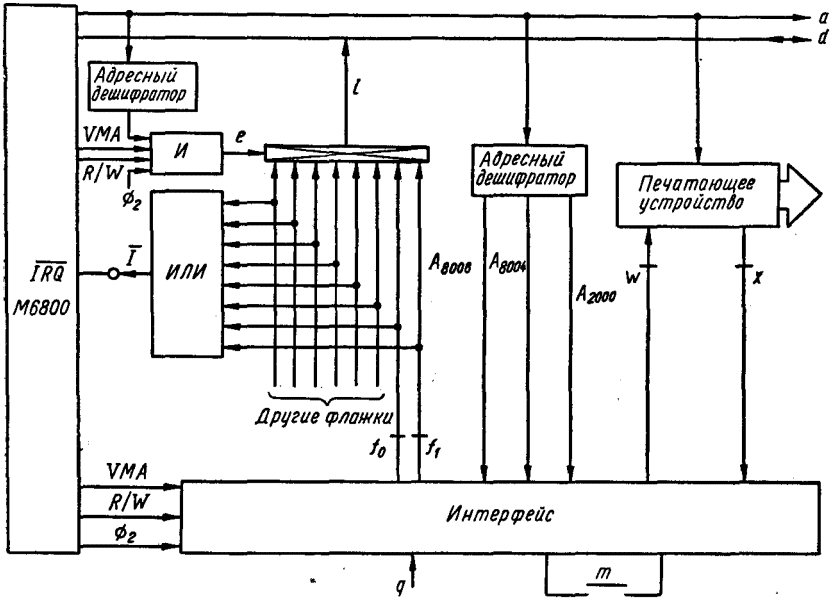


Рис. 5.30

варианте рассматривается последовательный метод опроса, описанный в 5.3. Структурная схема показана на рис. 5.30. Из нее видно, что интерфейс должен генерировать следующие сигналы:

1) два сигнала флажков — f_0 и f_1, вызывающих программы счета и печати;

2) сигнал разрешения *e* для порта ввода/вывода;

3) сигнал *w* для запуска печатающего устройства.

Как и ранее, в данной системе используется только два флажка. Это позволяет для формирования сигнала прерывания и адреса прерывающего устройства применять один вентиль *И* (см. 5.2). Кроме того, не требуются сигналы разрешения/запрета для этих флажков, и поэтому они могут быть реализованы на двух JK-триггерах, как показано на рис. 5.31. Импульс на шине *q* устанавливает триггер счетчика f_0, аналогично импульс на шине *m* устанавливает триггер печати f_1. Если для обнуления флажков

используются адреса 2000 и 8006, то уравнения сигналов сброса принимают следующий вид:

$$\overline{f_0} = \mathrm{VMA}\,\overline{\mathrm{R/W}}\,A_{2000}\Phi_2;$$

$$\overline{f_1} = \mathrm{VMA}\,\overline{\mathrm{R/W}}\,A_{8006}\Phi_2.$$

Вентили *И—НЕ*$_1$ и *И—НЕ*$_2$ на рис. 5.31 формируют эти сигналы.

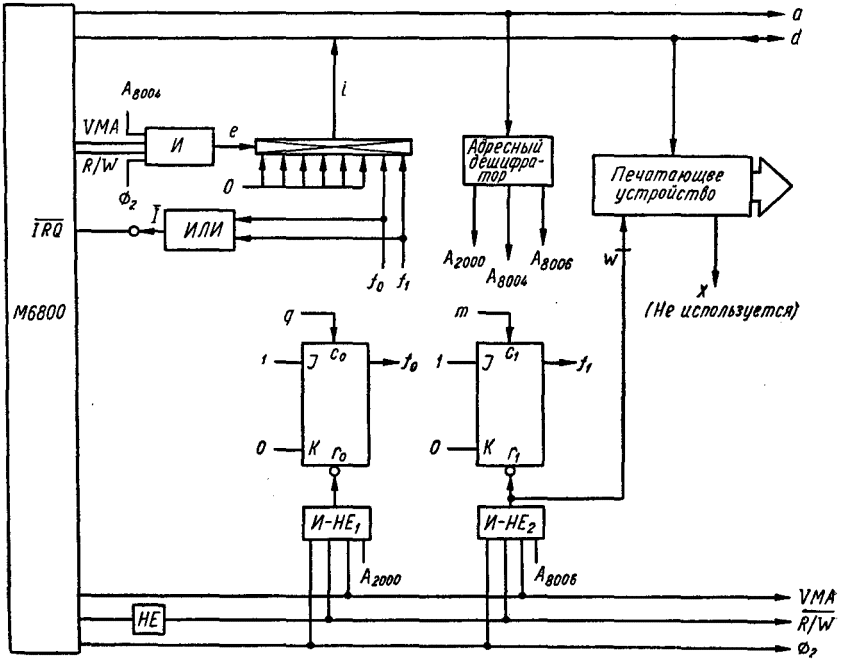


Рис. 5.31

В этой системе возможно обращаться к печатающему устройству, сбрасывая флажок f_1. Так как активизация печатающего устройства и сброс флажка происходят по спаду сигнала, то эти две операции можно объединить.

Обратите внимание на совпадение полученных схем для *In8080* и *Mc6800*.

Шаг 5 — проектирование программного обеспечения. Структурная схема алгоритма приведена на рис. 5.32. Непосредственно из этого рисунка, руководствуясь набором команд микропроцессора *Mc6800* (рис. 3.31), получаем шестнадцатиричный листинг служебной программы (табл. 5.2).

Задача 2. Интерфейс для распечатки содержимого ОЗУ. Разработать и реализовать интерфейс между 8-битовым микропроцессором и цифровым печатающим устройством, используя

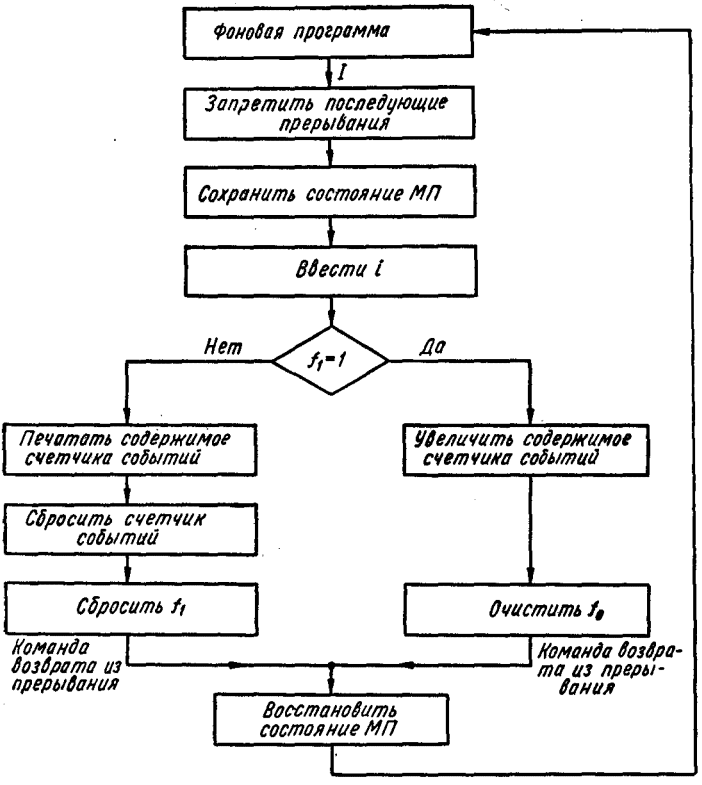


Рис. 5.32

режим прерывания. Реализовать интерфейс для 1) *In 8080*, 2) *Mc6800*.

Доступные адреса ввода/вывода:
для *In 8080* — 004_8, 005 и 006;
для *Mc6800*—2000_{16}, 4000 и 8004.

Решение для *In 8080*.

Шаг 1 — формирование технического задания. Цель проектирования — обеспечение программиста возможностью пересылки в режиме прерывания блоков данных, расположенных в последовательных ячейках ОЗУ, в приемник (рис. 5.33).

Рис. 5.33

Шаг 2 — внешнее проектирование. Структурная схема системы прерываний на базе *In 8080* показана на рис. 5.17, а сигналы

Таблица 5.2

Подпрограмма обработки прерываний в счетчике событий на Mc6800

	16-й адрес		16-й код	Мнемокод	Комментарии
	00	50	B6	LDA A	Ввод адреса из шифратора флажков
		51	80		
		52	04		
		53	87	RAR A	Циклический сдвиг вправо *A*
		54	25	BCS L1	Переход к подпрограмме счета, если флажок переноса установлен
		55	07		
		56	F6	LDA B	Переслать содержимое счетчика в *B*
		57	00		
		58	FF		
		59	F7	STA B	Печатать и очистить флажок f_1
		5A	80		
		5B	06		
		5C	3B	RTI	Возврат
L1:		5D	7C	INC	Увеличить содержимое счетчика
		5E	00		
		5F	FF		
		60	F7	STA B	Очистить флажок f_2
		61	20		
		62	00		
		63	3B	RTI	Возврат

сопряжения микропроцессора — на рис. 4.7. Характеристики сопряжения печатающего устройства показаны на рис. 5.24.

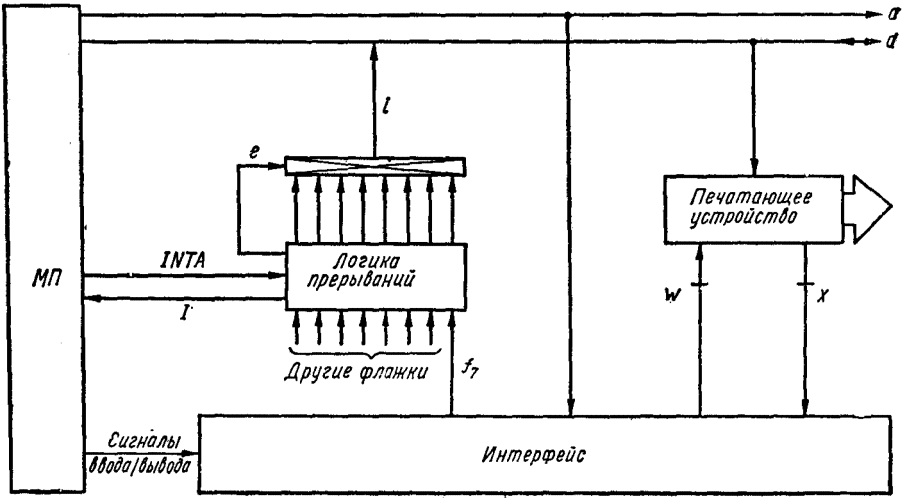


Рис. 5.34

Шаг 3 — системное проектирование. Структурная схема разрабатываемой системы показана на рис. 5.34, а алгоритм ее функционирования — на рис. 5.35. При каждом программном прерывании предназначенный для печати символ пересылается из ячей-

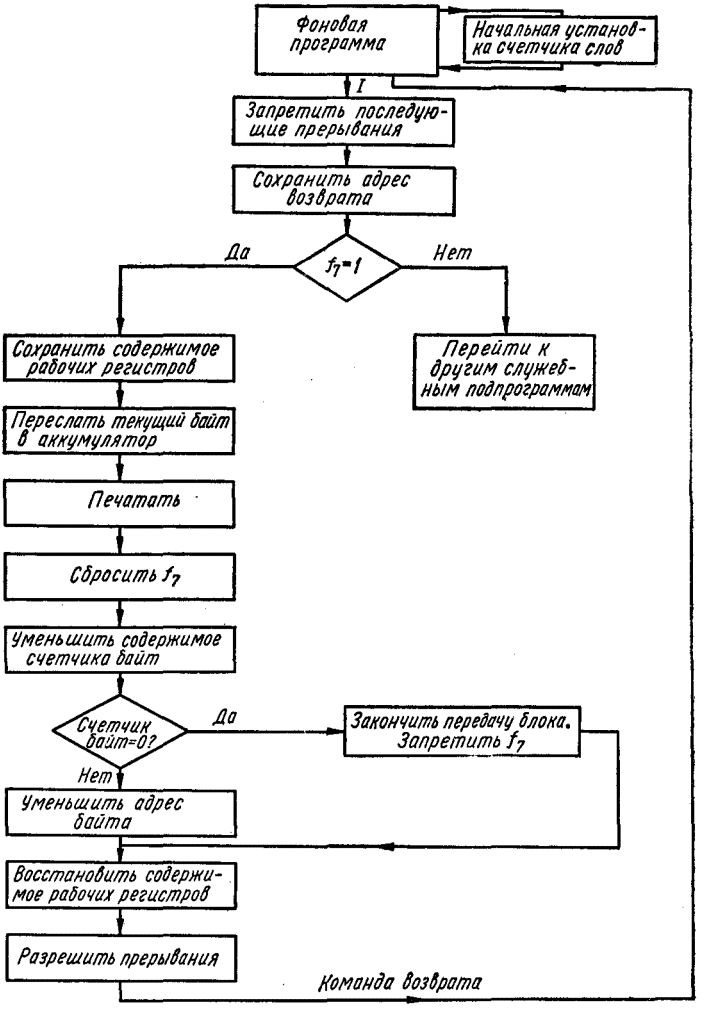


Рис. 5.35

ки ОЗУ в аккумулятор, а из аккумулятора — на шину данных, причем одновременно активизируется печатающее устройство. После того как символ отпечатан и печатающее устройство выставило флажок готовности, генерируется следующее прерывание путем установки флажка f_7.

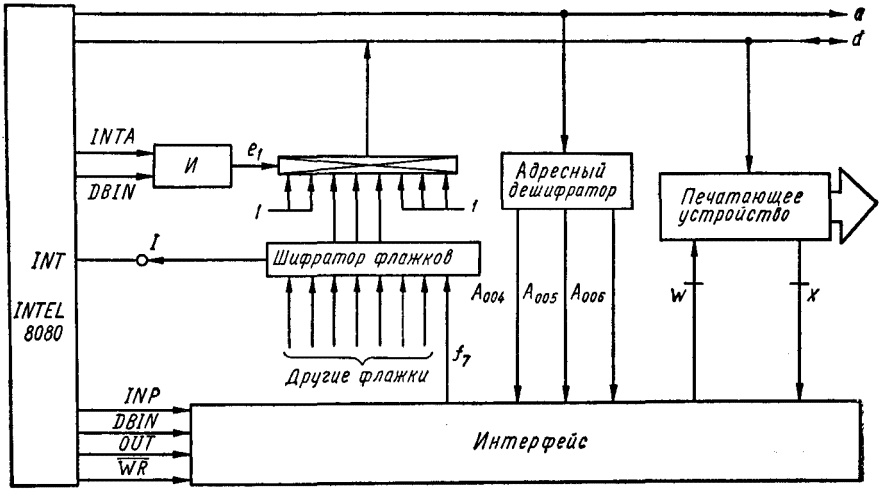


Рис. 5.36

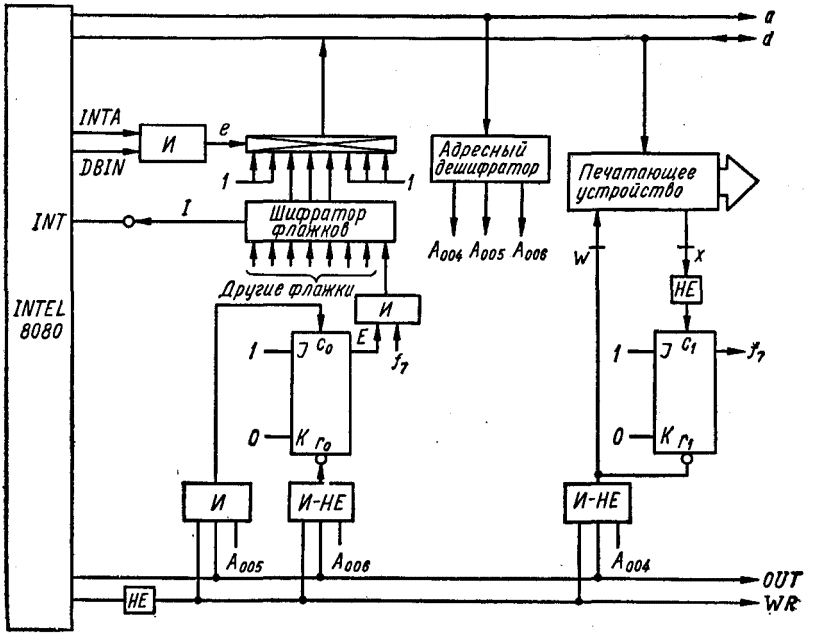


Рис. 5.37

Когда будет отпечатан последний символ из блока, программно запрещается установка флажка прерывания и дальнейшая печать запрещается. Передача следующего блока инициируется программистом путем разрешения прерываний.

Шаг 4 — проектирование аппаратурной части. Структурная схема для *In 8080* изображена на рис. 5.36. Сигналы, генерируемые интерфейсом, следующие: 1) импульс печати w, 2) флажок прерывания f_7.

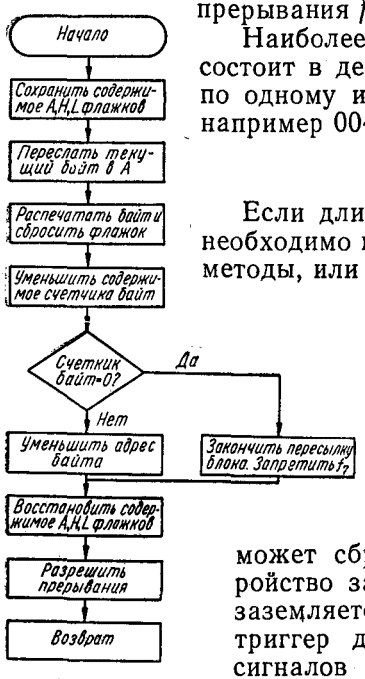


Рис. 5.38

Наиболее простой метод генерации сигнала w состоит в декодировании команды ввода/вывода по одному из доступных аресов ввода/вывода, например 004_8. В этом случае

$$w = \overline{\text{OUT WR } A_{004}}.$$

Если длительность сигнала w меньше 1 мкс, необходимо продлить его, используя стандартные методы, или добавить к печатающему устройству логическую схему, чувствительную к фронтам импульсов, используя методы, описанные в приложении 1.

Флажок f_7 должен устанавливаться, когда печатающее устройство выставляет сигнал готовности, т. е. когда сигнал его состояния принимает значение «1». Он может сбрасываться, когда печатающее устройство запрашивается, т. е. когда сигнал w заземляется. Если использовать синхронный триггер для формирования f_7, то уравнения сигналов его синхронизации и сброса будут иметь вид (см. 5.2)

$$c_1 = \overline{x}; \quad r_1 = w.$$

Схема триггера флажка показана на рис. 5.37. Читатель помнит, что триггеры переключаются по спаду синхросигнала, а по входу сброса реагируют на нулевой уровень.

Кроме того, интерфейс должен обеспечивать программное разрешение и запрет выдачи флажка прерывания. Этого можно достичь с помощью триггера, сбрасываемого и устанавливаемого программно. Если для этих целей использовать восьмеричные адреса 005 и 006, то соответствующие уравнения примут вид

$$c_0 = \text{OUT WR } A_{005}; \quad r_0 = \text{OUT WR } A_{006}.$$

Схема реализации этих уравнений показана на рис. 5.37.

Шаг 5 — проектирование программного обеспечения. Структурная схема алгоритма показана на рис. 5.38. Из этой схемы и

системы команд (рис. 3.28) получим восьмеричный и шестнадцатиричный листинги программы (табл. 5.3).

Таблица 5.3

Программа распечатки содержимого ОЗУ по прерыванию для *In8080*

8-й адрес		8-й код	16-й код	Мнемокод	Комментарии
003	060	365	F5	PUSH PSW	Сохранить содержимое рабочих регистров
	1	345	E5	PUSH H	
	2	032	1A	LHLD	Загрузить в регистры HL адрес следующего байта (содержимое ячеек 003-056/7)
	3	056	2E		
	4	003	03		
	5	276	7E	MOVA, M	Переслать байт в *A*
	6	323	D3	OUT	Вывести символ и очистить флажок
	7	004	04		
	070	053	2D	DCX H	Уменьшить содержимое H, L и сохранить его в ячейках 003-056/7 Это есть адрес следующего байта
	1	042	22	SHLD	
	2	056	2E		
	3	003	03		
	4	041	21	LXI H	Загрузить в H, L адрес ячейки памяти, где хранится число байт
	5	055	2D		
	6	003	03		
	7	065	35	DCR M	Уменьшить число байт
	100	302	C2	INZ	Если число байт не равно 0, перейти к L1
	1	105	45	L1	
	2	003	03		
	3	323	D3	OUT	Запретить f_7, если число байт равно 0
	4	006	06		
L1:	5	341	E1	POP H	Восстановить содержимое рабочих регистров
	6	361	F1	POP PSW	
	7	373	F8	EI	Разрешить прерывания
	10	311	C9	RET	Возврат

Решение для *Mc6800.*

Шаг 1
Шаг 2 } То же самое, что и для Мп *In8080.*
Шаг 3

Шаг 4 — проектирование аппаратурной части. Из структурной схемы на рис. 5.39 видно, что интерфейс формирует следующие сигналы: 1) сигнал печати w, 2) флажок прерывания f_7.

Если печатающему устройству присвоить шестнадцатиричный адрес 8004, то

$$w = A_{8004}\,\mathrm{VMA}\,\overline{\mathrm{R/W}}\,\Phi_2 \text{ (см. рис. 5.40).}$$

Если длительность сигнала w менее 1 мкс, то необходимо продлить ее стандартными методами либо ввести в состав печатающего устройства логическую схему, чувствительную к фронтам импульсов (см. приложение 1).

Флажок f_7 должен устанавливаться, когда печатающее устройство выставляет сигнал готовности, т. е. когда его сигнал состояния x становится равным «1». Флажок может быть сброшен, когда активизируется печатающее устройство, т. е. когда

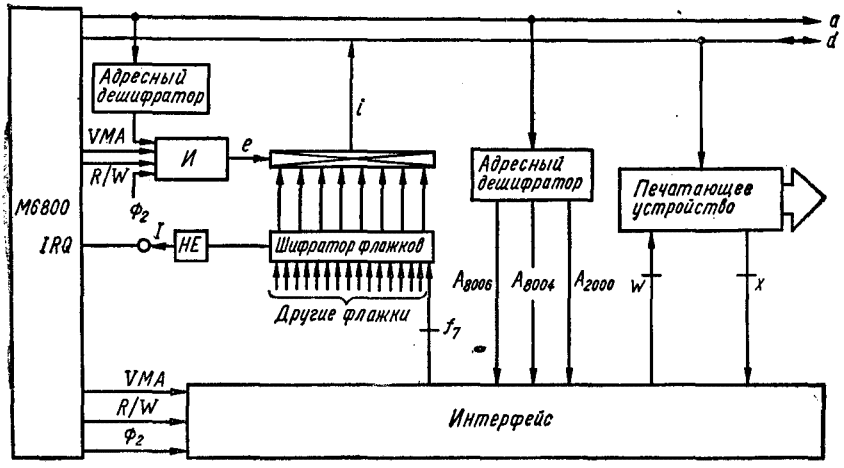


Рис. 5.39

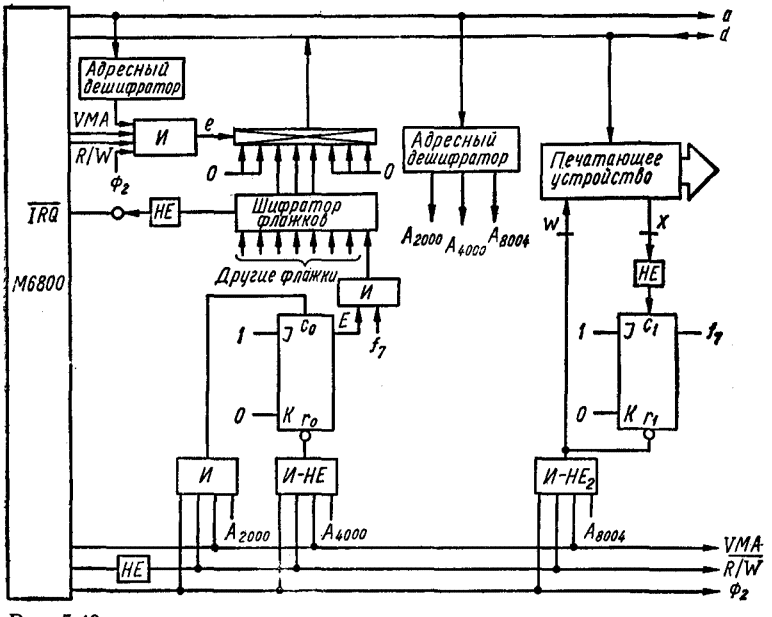


Рис. 5.40

контакт w заземляется. Если использовать синхронный триггер для формирования f_7, то уравнения сигналов синхронизации и сброса принимают вид (см. 5.2)

$$c_1 = \overline{x}; \quad r_1 = w.$$

Схема триггера показана на рис. 5.40. Напоминаем читателю, что применяемые здесь триггеры переключаются по спаду синхроимпульса, а по входу сброса реагируют на нулевой уровень.

Кроме того, интерфейс должен обеспечивать программное разрешение и запрет флажка. Наиболее простым методом для реализации этой возможности является применение триггера, устанавливаемого и сбрасываемого программно. Если для этой цели использовать шестнадцатиричные адреса 2000 и 4000, то уравнения сигналов синхронизации и сброса примут вид

$$c_0 = A_{2000}\,\mathrm{VMA}\,\overline{\mathrm{R/W}}\,\Phi_2;$$

$$\overline{r_0} = A_{4000}\,\mathrm{VMA}\,\overline{\mathrm{R/W}}\,\Phi_2.$$

Выход триггера флажка объединяется по схеме *И* с сигналом *E* для формирования сигнала f_7, как показано на рис. 5.40.

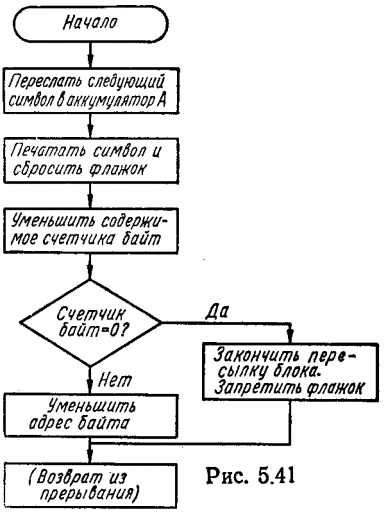


Рис. 5.41

Шаг 5 — проектирование программного обеспечения. Структурная схема алгоритма показана на рис. 5.41. Из этой схемы и системы команд МП из рис. 3.30 получим шестнадцатиричный листинг программы (табл. 5.4).

Таблица 5.4

Программа распечатки содержимого ОЗУ по прерыванию для Мс6800

16-й адрес		16-й код	Мнемокод	Комментарии
00	5F	*n*		Запомнить длину блока в ячейке 005F
	60	B6	LDA A	Переслать следующий символ в аккумулятор
	61	m_H		
	62	m_L		
	63	B7	STA A	Распечатать символ и очистить флажок f_7
	64	80		
	65	04		
	66	7A	DEC *n*	Уменьшить длину блока
	67	00		
	68	5F		
	69	27	BEQ L1	Если длина блока равна 0, перейти к L1
	6A	04		
	6B	7A	DEC m_L	Уменьшить содержимое указателя следующего символа
	6C	00		
	6D	62		
	6E	3B	RTI	Возврат
L1:	6F	B7	STA A	Запретить флажок f_7
	70	40		
	71	00		
	72	3B	RTI	Возврат

5.9. СПИСОК ЛИТЕРАТУРЫ

1. Zissos D., Duncan F. G. Digital Interface Design. Oxford University Press, 1973.
2. Zissos D. Problems and Solutions in Logis Design Oxford University Press, 1976.
3. INTEL 8080 Microprocessor Systems User's Manual, September, 1975.
4. M6800 Microprocessor Applications Manual. Motorola, 1975.

ГЛАВА 6

СИСТЕМЫ ПРЯМОГО ДОСТУПА К ПАМЯТИ (ПДП)

В этой главе рассматривается реализация микропроцессорных систем прямого доступа к памяти, т. е. систем, которые позволяют пересылать данные непосредственно между периферийными устройствами и памятью. Используемая методика проектирования и ее шаги описаны в гл. 2 (2.9 и 2.10 соответственно).

6.1. БАЗОВЫЕ ОПРЕДЕЛЕНИЯ

В рассматриваемых ранее методах пересылка данных между микропроцессором и внешними устройствами производилась через микропроцессор (рис. 6.1, *а*). Для этой цели требовалось несколько команд для передачи каждого байта. Например, при работе в режиме прерывания, как показано в гл. 5, необходимы следующие операции.

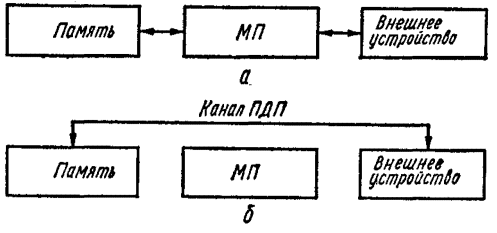


Рис. 6.1

1. Запретить дальнейшие прерывания, если это не выполняется автоматически.
2. Запомнить информацию о возврате.
3. Идентифицировать источник прерывания.
4. Обслужить запрос.
5. Очистить флажок.
6. Восстановить информацию о возврате.
7. Разрешить прерывание.
8. Возвратиться в прерванную программу.

Для обработки больших блоков данных такая процедура может потребовать значительное время. Более того, если скорость передачи информации с внешнего источника больше скорости обмена микропроцессора с памятью, возможны потери информации.

В системах, в которых установлена прямая связь между внешним устройством и памятью, как показано на рис. 6.1, *б*, проблема потери информации не возникает, так как в них можно передать байт информации между памятью и внешним устройством за один цикл памяти. Вопреки всеобщему убеждению проектирование и реализация такой системы весьма просты. Как мы увидим далее в этой главе, аппаратурный интерфейс несложен, а требуемое программное обеспечение вообще минимально — несколько команд на каждый пересылаемый блок.

Продолжительность времени прямого доступа к памяти исчисляется в циклах памяти. Цикл памяти — промежуток времени, в течение которого формируются требуемые сигналы чтения или записи. Так как циклы памяти эффективно используются для микропроцессорных операций в момент возникновения запроса прямого доступа, то цикл прямого доступа к памяти часто называют захваченным циклом, а саму операцию — захватом цикла. Все микропроцессоры имеют возможность прямого доступа к памяти (сокращенно ПДП), т. е. возможность прямой передачи данных между внешним устройством и памятью системы (рис. 6.1, *б*).

6.2. СИСТЕМА ПДП

Структурная схема системы ПДП показана на рис. 6.2. Она проектировалась как пример системы общего назначения, с тем

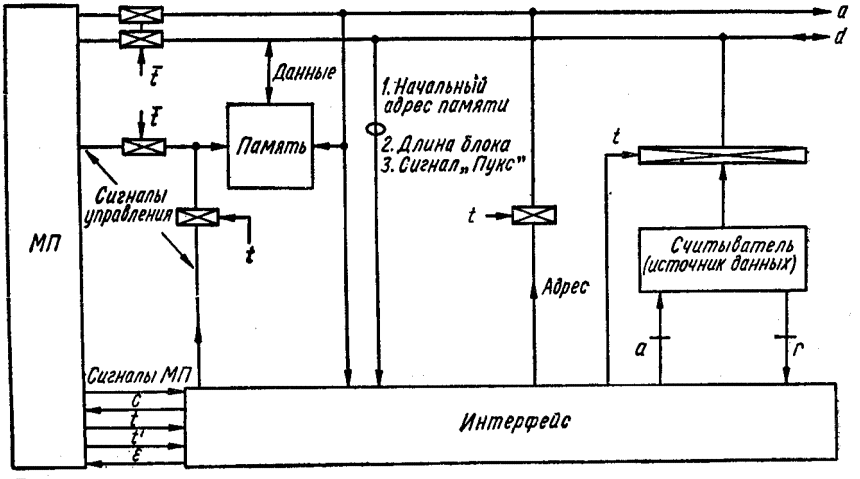


Рис. 6.2

чтобы удовлетворить любому типу периферийного устройства и любому типу микропроцессора. Алгоритм ее функционирования показан на рис. 6.3 и описан ниже.

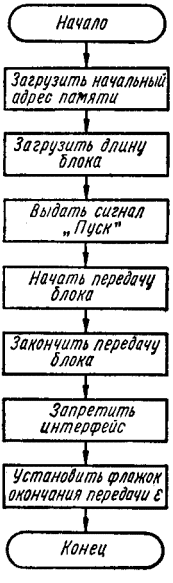


Рис. 6.3

1. Программист обращается к интерфейсу ПДП. Осуществляется это использованием команды ввода/вывода для выдачи на интерфейс первоначального адреса памяти и длины блока. Таким образом, интерфейсу ПДП ставятся в соответствие определенные адреса ввода/вывода, по которым программист имеет к нему доступ.

2. Далее выводится на интерфейс третья величина — сигнал запуска. Этот сигнал используется для начала пересылки блока, которая затем протекает автономно, без участия программиста.

3. Когда передача блока закончена, интерфейс устанавливает флажок ε, с тем чтобы сообщить программисту, что блок передан.

4. Программист подтверждает прием состояния флажка выполнением команды ввода/вывода, сбрасывающей его.

Сигналы c, t и t' описаны в следующем параграфе.

6.3. ИНТЕРФЕЙС ПДП

Проектирование систем ПДП, как уже отмечалось, не представляет особой сложности для читателя, знакомого с принципами логического проектирования, описанными в гл. 1.

Интерфейс ПДП проще разрабатывать, если рассматривать его состоящим из двух отдельных блоков: интерфейс *1* и интерфейс *2* (рис. 6.4). Интерфейс *1* предназначен для обработки инициирующей информации, вырабатываемой микропроцессором, и для формирования сигналов *«СТАРТ»* и *«СТОП»* для интерфейса *2*, который, в свою очередь, предназначен для управления передачей данных между памятью и внешними устройствами.

Алгоритм функционирования каждого из интерфейсов описан ниже.

Интерфейс 1. Структурная схема интерфейса 1 показана на рис. 6.5. Он состоит из двух каскадно соединенных счетчиков, пяти логических элементов и триггера. Счетчики предназначены для запоминания первоначального адреса памяти и длины блока. Загрузка их производится следующим образом. Программно засылается начальный адрес ячейки памяти в аккумулятор и затем выполняется команда ввода/вывода по адресу A_p. При этом формируется импульс ввода/вывода на входах загрузки каждого счетчика, по которому содержимое аккумулятора засылается в

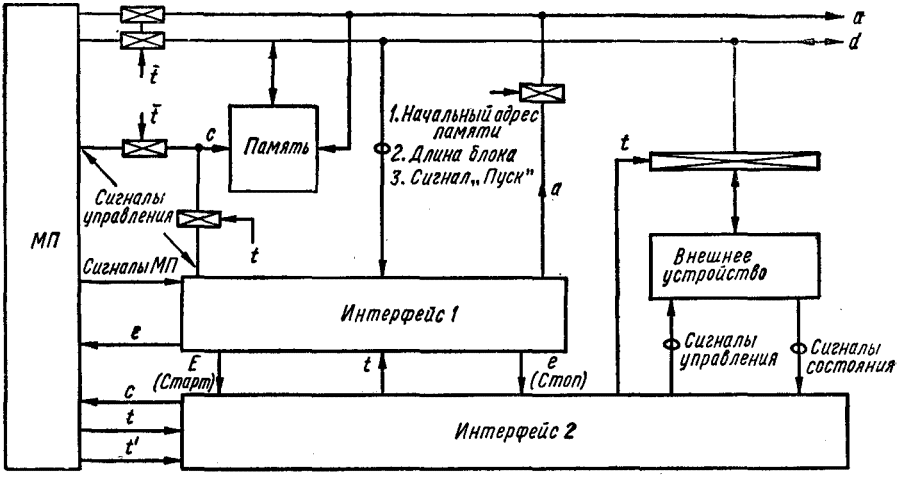

a
d
t
Память
c
1. Начальный адрес памяти
2. Длина блока
3. Сигнал „Пуск"
a
t
Сигналы управления
t
МП
Сигналы МП
Внешнее устройство
Интерфейс 1
e
Сигналы управления
Сигналы состояния
E (Старт)
t
e (Стоп)
c
t
t'
Интерфейс 2

Рис. 6.4

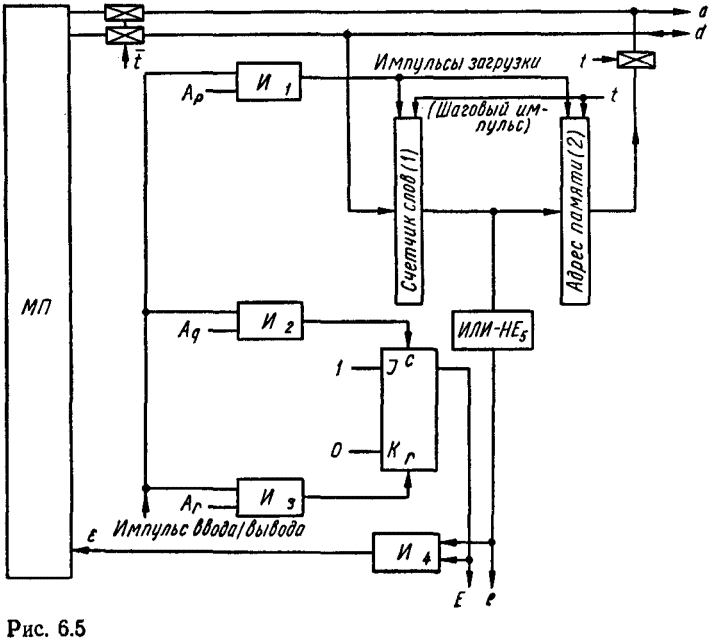

a
d
t
Импульсы загрузки
t
t
(Шаговый импульс)
A_p
И 1
Счетчик слов (1)
Адрес памяти (2)
МП
A_q
И 2
ИЛИ-НЕ$_5$
1
J
c
0
K
r
A_r
И 3
e
Импульс ввода/вывода
И 4
E
e

Рис. 6.5

счетчик *1*. В то же самое время, благодаря каскадному соединению счетчиков, прежнее содержимое счетчика *1* переписывается в счетчик *2*. Далее программным путем в аккумулятор загружается величина длины блока и опять выполняется та же самая команда ввода/вывода. При этом содержимое счетчика *1* (начальный адрес памяти) переносится в счетчик *2*, а величина длины блока загружается из аккумулятора в счетчик *1*.

Далее программа отрабатывает следующую команду ввода/вывода по адресу A_q(рис. 6.5). При этом формируется импульс

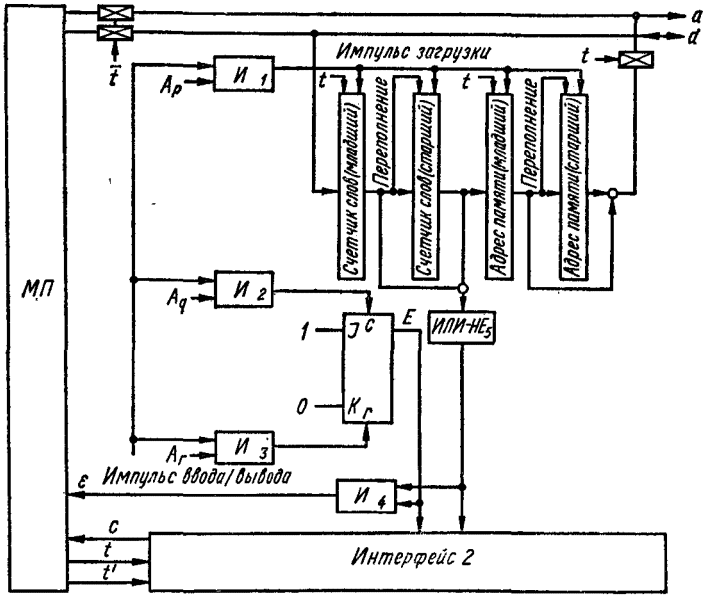


Рис. 6.6

на входе синхронизации JK-триггера. В связи с тем, что на вход *J* триггера постоянно подана «1» и триггер вначале сброшен сигналом системного сброса, то он устанавливается в «1» и формируется сигнал *E*. Этот сигнал, как будет показано далее, является приказом интерфейсу *2* на запуск передачи блоков данных путем запуска цикла ПДП каждый раз, как только периферийное устройство выставит сигнал готовности.

Когда цикл ПДП закончен, сигнал *t* стансвится равным «1». Этот сигнал используется для увеличения содержимого обоих счетчиков на единицу (рис. 6.5).

По окончании передачи блока данных в счетчике *1* записаны нули, что индицируется логической «1» на выходе элемента *ИЛИ—НЕ*. Выходной сигнал этого элемента, обозначенный буквой *e* на рис. 6.5, используется интерфейсом *2* для окончания пересылки блока. В то же время он объединяется по схеме *И*

с сигналом E для выработки сигнала *«конец передачи»* ε, который, как было указано выше, информирует программиста об окончании передачи блока. Программист подтверждает прием флажка ε выполнением команды ввода/вывода по адресу A_r. По этой команде после дешифрации элементом $И_3$ (рис. 6.5) формируется импульс на вход сброса JK-триггера, благодаря чему выключается сигнал ε.

На рис. 6.6 показана структурная схема интерфейса *1* для микропроцессора с 8-битовой шиной данных и 16-битовой шиной адреса. Для упрощения на этом рисунке не показано периферийное устройство. Смысл сигналов *c*, *t* и *t'* объяснен ниже.

Рис. 6.7

Интерфейс 2. Функция интерфейса *2*, как уже указывалось, состоит в передаче данных между памятью микро-ЭВМ и внешними устройствами. Для абстрагирования процесса проектирования от типа микропроцессора предположим, что используется некоторый микропроцессор, позволяющий выполнять захват цикла (см. 6.5).

Сигнал c. Переход от «0» к «1» этого сигнала переводит буферы микропроцессора в третье состояние для выполнения цикла ПДП. Таким циклом (см. 6.1) называется цикл времени, в течение которого генерируются необходимые сигналы чтения/записи памяти.

Сигнал t. Этот сигнал равен «1», когда микропроцессор находится в третьем состоянии, в противном случае $t=0$. Этот сигнал используется для определения продолжительности цикла ПДП.

Сигнал t'. Сигнал t' генерируется в течение действия сигнала t. Он используется как импульс чтения/записи для памяти.

6.4. ДВУХПРОВОДНОЙ ИНТЕРФЕЙС

В этом параграфе рассмотрим реализацию интерфейса *2* (рис. 6.4) между микропроцессором с возможностями захвата цикла, объясненными в предыдущем параграфе, и устройством

типа запрос/ответ (см. приложение 1), состоящего из двух проводов. Воспользуемся методикой проектирования, описанной в параграфе 1.9. Для упрощения первоначально не будем рассматривать сигналы E, e и t' (рис. 6.8).

Шаг 1 — характеристики ввода/вывода. Сигналы ввода/вывода показаны на рис. 6.9. Необходимые соотношения между

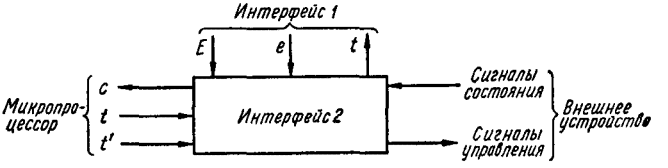


Рис. 6.8

входными и выходными сигналами описаны в предыдущем параграфе и характеризуются тем, что цикл ПДП запрашивается внешним устройством, когда оно готово читать или писать байт

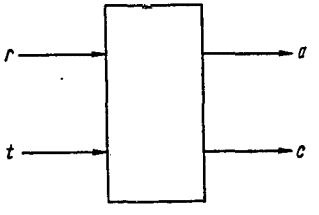


Рис. 6.9

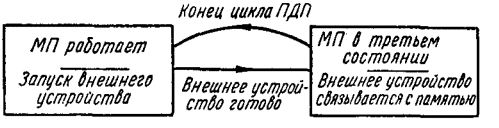


Рис. 6.10

памяти. Когда цикл ПДП выполняется, передается байт, по окончании передачи микропроцессор возобновляет функционирование, а внешнее устройство запускается (рис. 6.10).

Шаг 2 — внутренние характеристики. Внутренняя диаграмма состояний соответствующей схемы показана на рис. 6.11. Состоя-

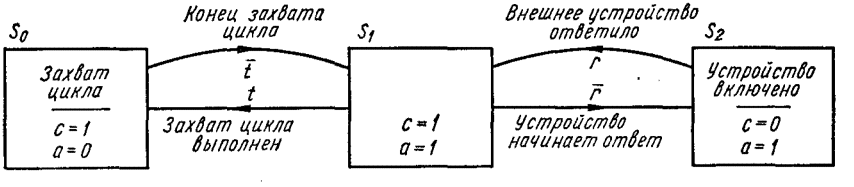


Рис. 6.11

ние S_0 поддерживается, пока микропроцессор находится в третьем состоянии, т. е. в течение цикла ПДП. В конце цикла ПДП, когда сигнал t переключается в «0», схема переходит в состояние S_1. Переход из S_0 в S_1 сопровождается изменением сигнала a из «0» в «1», благодаря чему запускается внешнее устройство. Когда внешнее устройство ответило или сигнал готовности r стал

нулевым, схема переходит в состояние S_2. В состоянии S_2 схема находится до тех пор, пока внешнее устройство не ответит полностью, т. е. пока сигнал $r=0$. Когда устройство ответило полностью, сигнал r становится равным «1», что вызывает переход

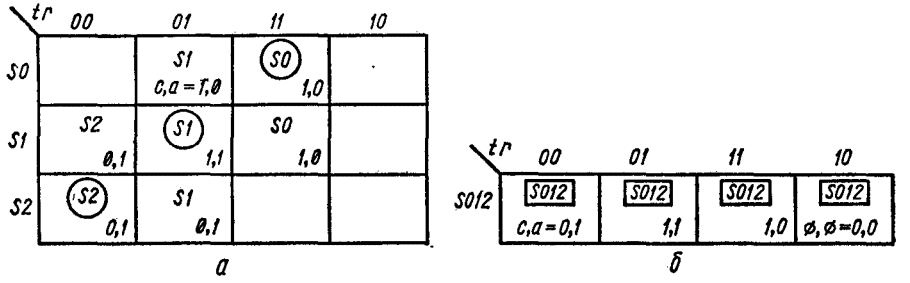


Рис. 6.12

системы в состояние S_1. При переходе из состояния S_2 в S_1 сигнал c изменяется от «0» до «1», запрашивая разрешение на захват цикла микропроцессора. Когда цикл захвачен ($t=1$), схема переходит в состояние S_0. В дальнейшем процесс повторяется.

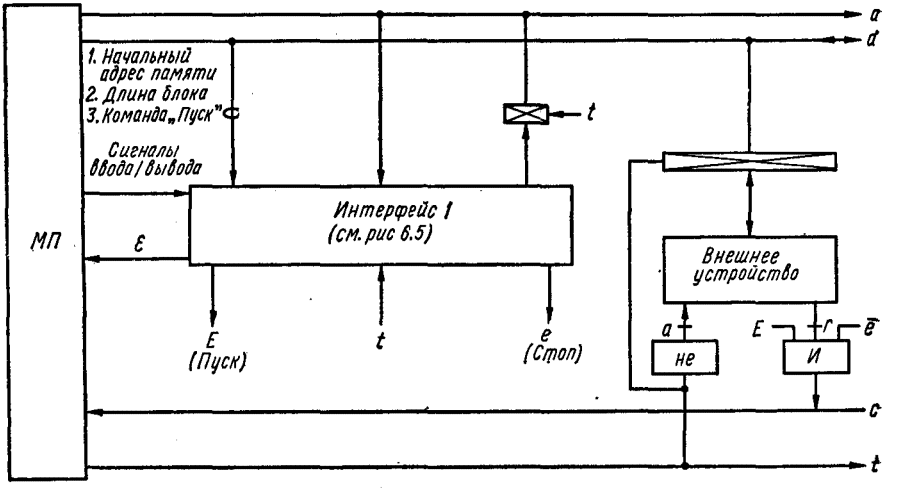


Рис. 6.13

Шаг 3 — минимизация числа состояний. На этом шаге, как уже указывалось, по диаграмме внутренних состояний составляется таблица состояний и используется методика минимизации Калдвелла. Таблица, полученная непосредственно из диаграммы состояний (рис. 6.11), показана на рис. 6.12, *а*. Применяя процедуру минимизации (см. 1.7), сливаем в таблице три строки в одну (рис. 6.12, *б*).

Шаг 4 — реализация схемы. Непосредственно из минимизированной таблицы состояний получаем уравнения

$$c = \bar{t}r + tr + (\bar{t}\bar{r}) = r; \tag{6.1}$$

$$a = \bar{t}\bar{r} + \bar{t}r + (\bar{t}\bar{r}) = t. \tag{6.2}$$

Вводя сигналы E и e, получаем

$$c = E\bar{e}r; \tag{6.3}$$

$$a = \bar{t}. \tag{6.4}$$

Таким образом, интерфейс ПДП между микропроцессором с захватом цикла и внешним устройством типа запрос/ответ состоит из двух проводов, как показано на рис. 6.13.

Из уравнений (6.3) и (6.4) следует, что передача данных запускается циклом ПДП и заканчивается запуском устройства.

6.5. ЛОГИЧЕСКАЯ СХЕМА ЗАХВАТА ЦИКЛА

Характеристики захвата цикла, определенные в предыдущем параграфе, в современных микропроцессорах отсутствуют. Таким образом, задача построения внешней логической схемы, реализующей функцию захвата цикла, выпадает на долю пользователя. Структурная схема показана на рис. 6.14,*а*. Процедура проектирования весьма проста.

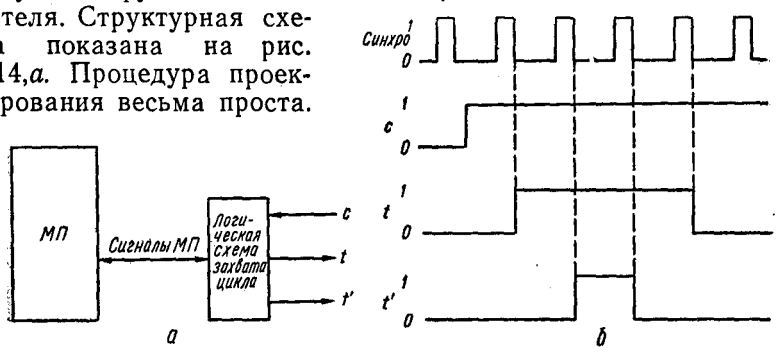


Рис. 6.14

Основная трудность состоит в правильной интерпретации по имеющимся описаниям функций выводов микропроцессора.

Как уже отмечалось выше, во всех случаях, кроме оговоренных специально, длительность цикла ПДП предполагается равной трем синхропериодам (рис. 6.14, *б*). Продемонстрируем процедуру разработки логики захвата цикла для микропроцессора *Intel 8080*.

Логическая схема захвата цикла для микропроцессора *Intel 8080*. Требуемые сигналы микропроцессора показаны на рис. 6.15, *а*. Логическая «1» на шине захвата *HOLD* (вывод *13*) отсоединяет адресную шину и шину данных. Сигнал ответа *HLDA* («*ПОД-*

ТВЕРЖДЕНИЕ ЗАХВАТА») выдается на вывод *21*. Этот сигнал принимает значение логической «1» через определенное время после появления фронта Φ_1 в машинном такте *Т3*. Шины дан-

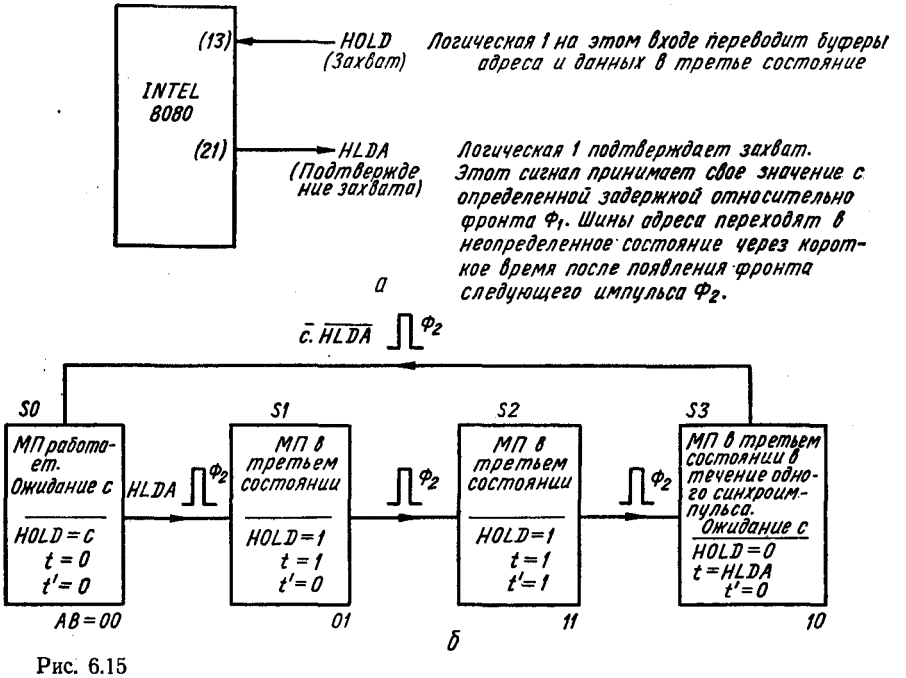


Рис. 6.15

ных и адресов переходят в третье состояние по истечении короткого промежутка времени после появления фронта следующего импульса Φ_2. Для более детального ознакомления с временны-

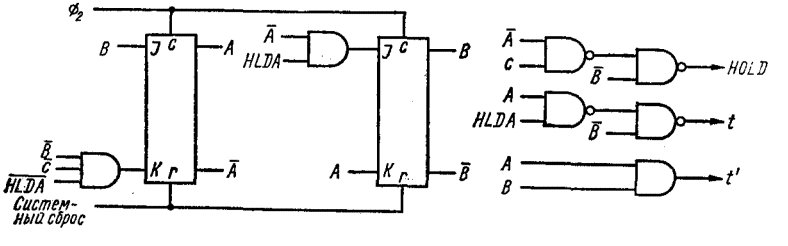


Рис. 6.16

ми диаграммами читателю рекомендуется обратиться к руководству по микропроцессору *In8080*.

На рис. 6.15, *б* показана внутренняя диаграмма состояний логической схемы, переводящей микропроцессор *In8080* в третье состояние примерно на 3 синхропериода. При этом необходимо

принять во внимание, что сигнал c не должен сбрасываться вплоть до момента появления сигнала $HLDA$. Непосредственно из этой диаграммы получаем уравнения:

$$S_A = S_1 = \overline{A}B, \text{ следовательно, } J_A = B;$$

$$R_A = S_3\overline{c}\,\overline{\text{HLDA}} = A\overline{B}\overline{c}\,\overline{\text{HLDA}}, \text{ следовательно } K_A = \overline{B}\overline{c}\,\overline{\text{HLDA}};$$

$$S_B = S_0\,\text{HLDA} = \overline{A}\overline{B}\,\text{HLDA}, \text{ поэтому } J_B = \overline{A}\,\text{HLDA};$$

$$R_B = S_2 = AB, \text{ поэтому } K_B = A;$$

$$\text{HOLD} = S_0c + S_1 + S_2 = \overline{A}\overline{B}c + \overline{A}B + AB = \overline{A}c + B;$$

$$t = S_1 + S_2 + S_3\,\text{HLDA} = \overline{A}B + AB + A\overline{B}\,\text{HLDA} = B + A\overline{B}\,\text{HLDA} = B + A\,\text{HLD}A;$$

$$t' = S_2 = AB.$$

Соответствующая элементарная схема показана на рис. 6.16.

6.6. ЗАДАЧИ И РЕШЕНИЯ

В этой главе методика проектирования демонстрируется на ряде задач с решениями. Внимание читателя обращается на тот факт, что, хотя для реализации используется микропроцессор

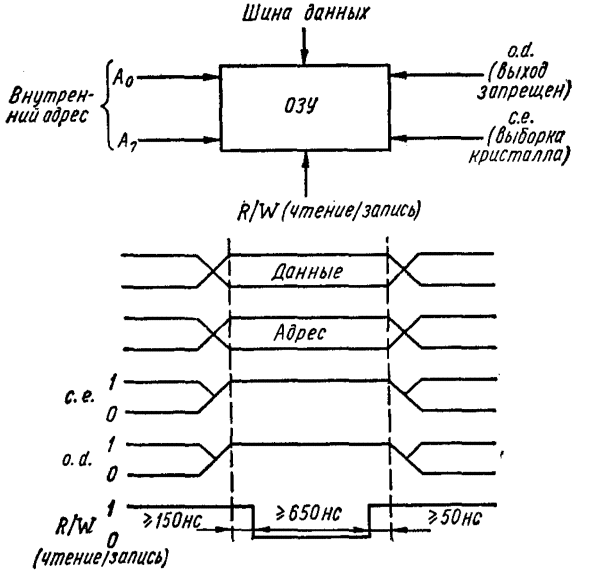


Рис. 6.17

In8080, данная методика применима ко всем типам микропроцессоров, имеющих или не имеющих логическую схему захвата цикла. Особо необходимо отметить, что первые три шага проектирования выполняются вообще без ссылки на применяемый тип микропроцессора.

Задача 1. Интерфейс считывателя с памятью. Разработать интерфейс ПДП между 8-рязрядным считывателем с перфоленты

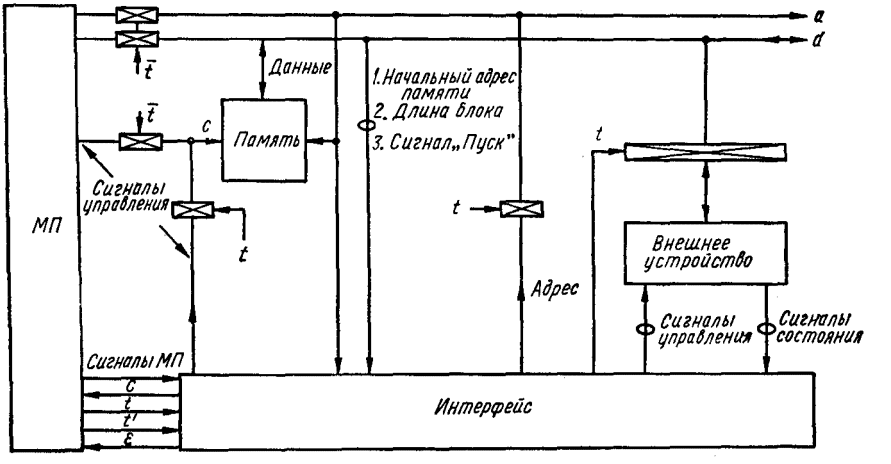


Рис. 6.18

и ОЗУ микропроцессорной системы. Доступные адреса ввода/вывода: A_{004}, A_{005}, A_{006}.

Решение.

Шаг 1 — формирование технического задания. Целью проектирования является разработка устройства, позволяющего пересылать блоки данных из источника непосредственно в ОЗУ микропроцессорной системы, используя принцип ПДП.

Шаг 2 — внешнее проектирование. Микропроцессор переходит в третье состояние на время трех синхропериодов по фронту импульса на контакте *c* (рис. 6.14). Принимая частоту синхроимпульсов, равную 0,5—1,5 МГц, получаем длительность сигнала *t*, равную 2—6 мкс, и *t'*, равную 667 нс — 2 мкс.

Цикл записи ОЗУ показан на рис. 6.17. Считыватель является устройством типа запрос/ответ (см. приложение 1).

Шаг 3 — системное проектирование. Структурная схема разрабатываемого устройства показана на рис. 6.18, а схема алгоритма функционирования его после начальной загрузки — на рис. 6.19.

Шаг 4 — проектирование аппаратурной части. Отправной точкой для разработки аппаратуры служит структурная схема, показанная на рис. 6.4,— общая структура системы ПДП. Реализация интерфейсов *1* и *2* показана на рис. 6.5 и 6.13. Если

предположить, что длина блока не превышает 256 слов, то соответствующая схема на базе *In8080* показана на рис. 6.20.

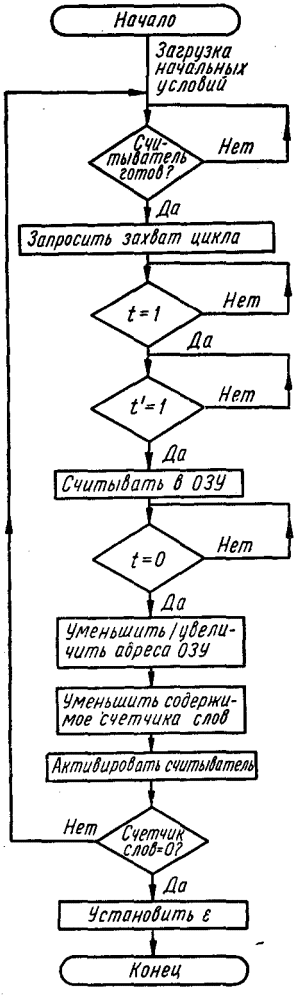


Рис. 6.19

Шаг 5 — проектирование программного обеспечения. Программа, необходимая для инициирования пересылки n символов со считывателя в последовательные ячейки ОЗУ через интерфейс рис. 6.20, имеет вид:

Загрузить аккумулятор начальным адресом ОЗУ (старшая половина)	Загрузка начального адреса ОЗУ
Вывод по адресу 004	
Загрузить аккумулятор начальным адресом ОЗУ (младшая половина)	
Вывод по адресу 004	
Загрузить аккумулятор длиной блока	Загрузка длины блока
Вывод по адресу 004	
Вывод по адресу 005	Начать передачу блока
Если ε = 1, выполнить служебную программу	
Вывод по адресу 006	Очистка флажка
EI	Разрешить прерывание
RET	Возврат

Задача 2. Модификация системы. Модифицировать решение в предыдущей задаче так, чтобы только данные, отмеченные признаками $v=1$, считывались в ОЗУ.

Решение.

Указанная модификация осуществляется путем подавления цикла ПДП и генерирования цикла чтения, когда $\overline{vr}=1$. (Введение переменной r в уравнение указывает, что данные передаются только тогда, когда $r=1$). Это достигается путем модификации сигналов c и a (рис. 6.20) следующим образом:

$$c = Er\overline{ev}; \quad a = tv\overline{r}.$$

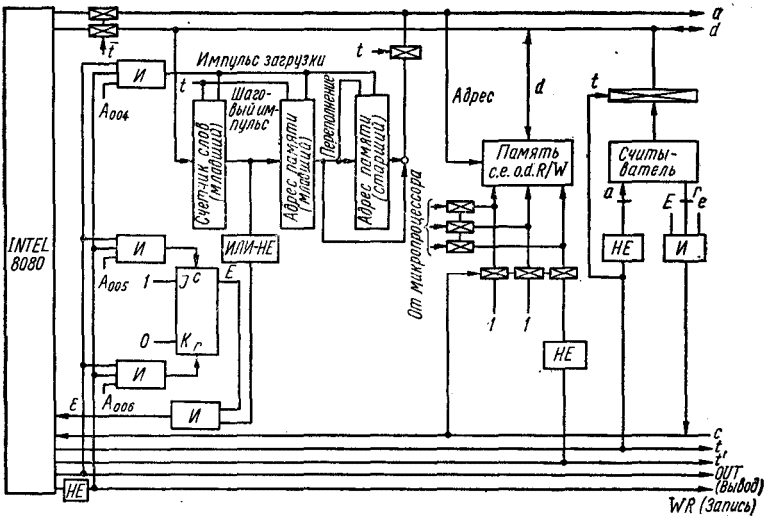


Рис. 6.20

ГЛАВА 7

СИСТЕМЫ ПРЯМОЙ ПЕРЕДАЧИ ДАННЫХ

В этой главе внимание уделено проектированию и реализации микропроцессорных систем прямой передачи данных (ППД), т. е. таких систем, в которых пересылка данных может производиться непосредственно между периферийными устройствами. Используемые методика и алгоритм проектирования описаны в 2.9 и 2.10 соответственно.

7.1. БАЗОВЫЕ ОПРЕДЕЛЕНИЯ

Рассмотрим следующую ситуацию. Необходимо получить распечатку некоторой информации, отперфорированной на перфоленте. Для этой цели воспользуемся фотосчитывателем и печатающим устройством микропроцессорной системы при условии, что микропроцессор нельзя занимать более, чем на несколько микросекунд для каждого акта печати.

Возможно следующие способы решения поставленной задачи:

1. Считывать каждый символ в аккумулятор, а затем распечатывать его (рис. 7.1). Этот метод требует нескольких команд для пересылки каждого байта со считывателя на печатающее

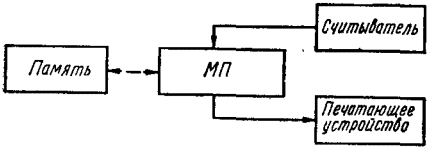


Рис. 7.1

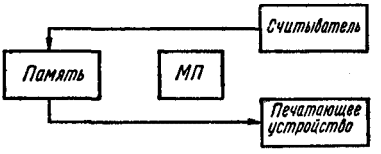


Рис. 7.2

устройство (подобная задача решалась в гл. 4, задача 3). В связи с тем, что время выполнения каждой команды обычно составляет несколько микросекунд, этот метод исключается.

2. Можно прочитать целиком содержимое ленты в память (ОЗУ), а затем отпечатать это содержимое, используя в каждом случае канал ПДП (рис. 7.2). Если на ленте имеется n символов, то потребуется $2n$ циклов ПДП и ряд команд для инициализации обоих интерфейсов, как объяснялось в предыдущей главе. Общее время, требуемое в данном случае, хотя и значительно меньше, чем в предыдущем способе, однако для современных микропроцессоров превышает 100 мкс. Для реализации метода требуется определенный объем памяти (ОЗУ), что не всегда имеется. Кроме того, необходимы два канала ПДП.

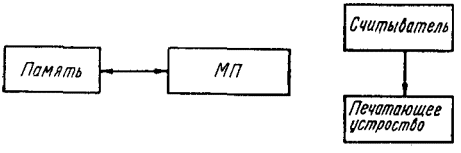


Рис. 7.3

3. Можно установить непосредственный канал связи между считывающим и печатающим устройствами (см. рис. 7.3). В такой системе данные передаются от источника к приемнику независимо от микропроцессора, благодаря чему удовлетворяются условия поставленной задачи.

Для упрощения понимания на рис. 7.1, 7.2 и 7.3 не показаны другие внешние устройства.

7.2. СИСТЕМЫ ППД

Структурная схема системы ППД показана на рис. 7.4. Аппаратурной части интерфейса присвоен некоторый адрес ввода/вывода, по которому к нему осуществляется доступ. Если пользователь хочет установить прямой канал передачи между парой или группой внешних устройств, он должен поступить следующим образом.

Во-первых, определить, доступно ли намеченное к использованию устройство. Эту информацию он получает опросом сигнала

b (рис. 7.4) — сигнала состояния, индицирующего занятость опрашиваемого устройства: $b=1$, если одно или более устройств заняты, в противном случае $b=0$. Если устройства свободны, необходимо выполнить команду ввода/вывода по адресу интерфейса. При этом интерфейс реагирует следующим образом: 1) изоли-

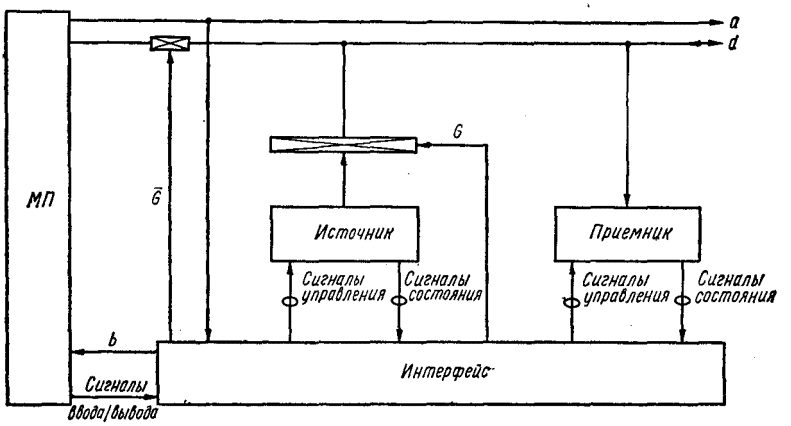


Рис. 7.4

рует от системы ту часть шины, которая соединяет запрашиваемые устройства; 2) изменяет сигнал b с «0» на «1»; 3) инициирует пересылку данных, которая далее протекает автоматически, т. е. без программного обслуживания. Когда полный блок данных передан, сигнал b становится равным «0», а участок шины возвращается в нормальное состояние.

7.3. ИНТЕРФЕЙС ППД

Разработка и реализация интерфейсов ППД производится по хорошо известным алгоритмам и не представляет труда для читателя, знакомого с методами логического проектирования [1].

Для устройств типа запрос/ответ (см. приложение 1) показано, что интерфейс может состоять из двух проводов [2]. Для полноты изложения повторим доказательство этого утверждения.

Начнем со структурной схемы, показанной на рис. 7.5, *а*. Ее функционирование описано схемой алгоритма, приведенной на рис. 7.5, *б*. Для простоты проектирования никакие внешние управляющие сигналы на данном этапе не рассматриваются.

С точки зрения реализации интерфейс представляет собой логическую схему с входными сигналами r_1 и r_2 и выходными сигналами a_1 и a_2 (рис. 7.6). Для реализации схемы используем алгоритм, описанный в параграфе 1.9.

Шаг 1 — внешние характеристики. Они показаны на рис. 7.5.

Шаг 2 — внутренние характеристики. Соответствующая диаграмма внутренних состояний показана на рис. 7.7. Функционирует она следующим образом.

Допустим, что устройство *1* активно, а устройство *2* нет (см. рис. 7.5, *а*). На диаграмме это состояние обозначено S_0. Система

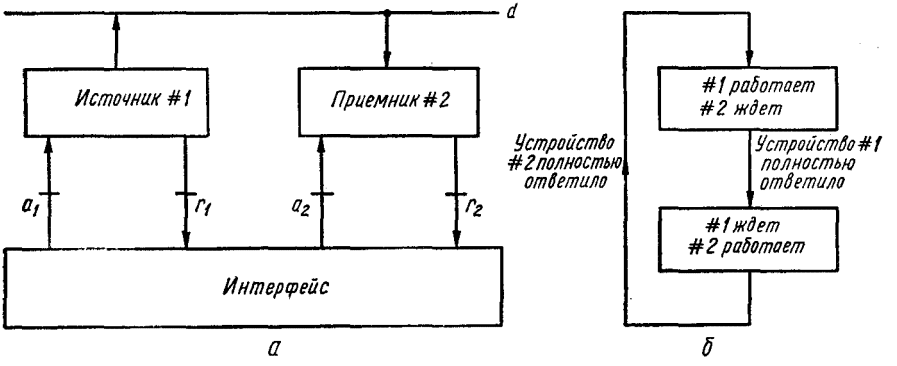


Рис. 7.5

находится в этом состоянии до тех пор, пока устройство *1* активно. Когда оно заканчивает работу, выставляя сигнал $r_1=1$, схема переходит в состояние S_1. Из рис. 7.7 видно, что в состоянии S_0 $a_2=0$, а в состоянии S_1 $a_2=1$, т. е. переход из S_0 в S_1 вызывает переход сигнала запроса a_2 из «0» в «1». Этот переход активизирует устройство 2. Когда r_2 становится равным 0, т. е. устройство *2* начинает работать, схема переходит в состояние S_2. Она остается в этом состоянии до тех пор, пока устройство *2* не закончит работу, т. е. r_2 не станет равным «1». Когда r_2 становится равным «1», схема переходит в состояние S_1. Так как a_1 равно «0» в состоянии S_2 и a_1 равно «1» в состоянии S_1, то переход из S_2 в S_1 активизирует устройство *1*. Когда это устройство отрабатывает приказ, о чем свидетельствует значение $r_1=0$, схема переходит в состояние S_0 и продолжает в нем находиться до тех пор, пока устройство *1* не освободится. Далее цикл повторяется.

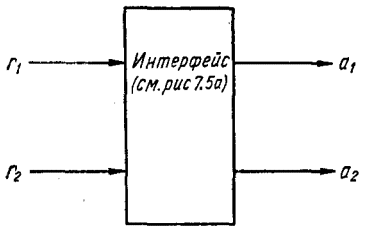


Рис. 7.6

Шаг 3 — минимизация числа состояний. Таблица состояний, соответствующая диаграмме состояний рис. 7.7, показана на рис. 7.8, *а*. Применяя методику минимизации числа состояний, описанную в параграфе 1.7, соединим три строки таблицы состояний в одну, как показано на рис. 7.8, *б*. В первой ячейке введем дополнительно состояние S_{012}, так как никакое другое состояние схема принять не может. Для данной ячейки выходными сигна-

лами служат ∅, ∅, принимающие нулевые (безразличные) значения.

Шаг 4 — реализация схемы. Непосредственно из минимизированной таблицы состояний (рис. 7.8, *б*) получаем следующие уравнения:

$$a_1 = \bar{r}_1 r_2 + r_1 r_2 + (\bar{r}_1 \bar{r}_2) = r_2; \tag{7.1}$$

$$a_2 = r_1 r_2 + r_1 \bar{r}_2 + (\bar{r}_1 \bar{r}_2) = r_1. \tag{7.2}$$

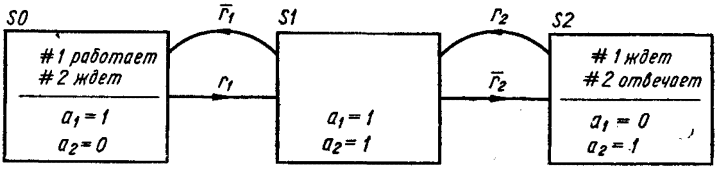


Рис. 7.7

Так как дополнительное произведение $\bar{r}_1 \bar{r}_2$ не будет использовано в окончательном выражении для сигналов a_1 и a_2, то $a_1 = a_2 = 0$ в первой ячейке рис. 7.8, *б*.

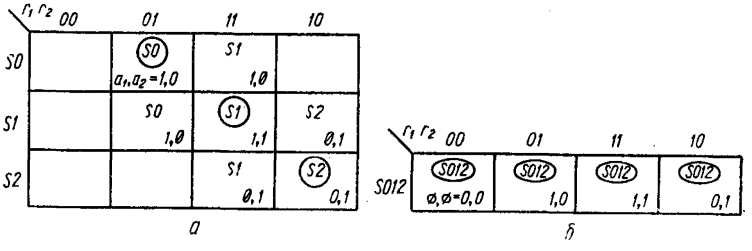


Рис. 7.8

Будем называть уравнения (7.1) и (7.2) простейшими интерфейсными уравнениями. Реализация их соответствует двум проводникам, показанным на рис. 7.9.

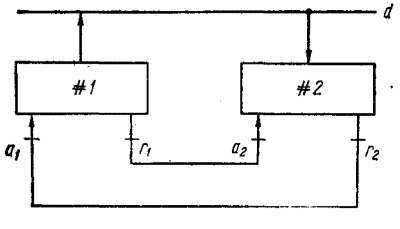


Рис. 7.9

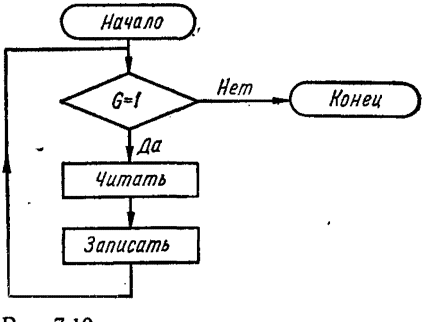


Рис. 7.10

Управление работой/остановом. Очевидно, чтобы избежать потери информации, пересылка данных должна начинаться операцией чтения, а заканчиваться операцией записи (рис. 7.10).

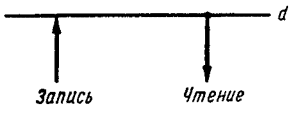


Рис. 7.11

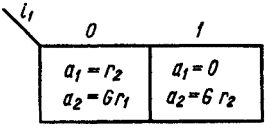


Рис. 7.12

Сами операции чтения/записи определены на рис. 7.11. Используем сигнал G для определения активного ($G=1$) и неактивного ($G=0$) состояний системы (рис. 7.12). Непосредственно из диаграммы получаем

$$a_1 = \bar{G} r_2 + G r_2 = r_2;$$

$$a_2 = G r_1.$$

Соответствующая схема показана на рис. 7.13.

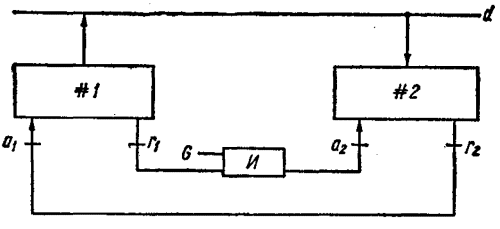


Рис. 7.13

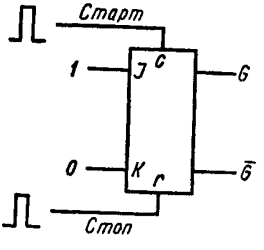


Рис. 7.14

Если сигналы старт/стоп являются импульсными, можно использовать JK-триггер (рис. 7.14) для генерации сигнала G. Возможны и другие варианты.

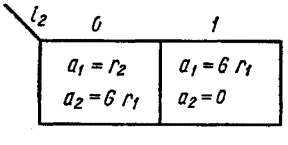


Рис. 7.15

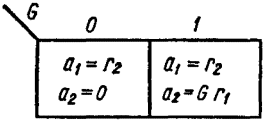


Рис. 7.16

Операция подавления чтения ($i_2 = 1$). В рассматриваемой системе операция подавления чтения реализована путем запрета чтения ($a=0$) и инициирования вместо нее операции записи ($a_1 = G r_1$), как показано на рис. 7.15. Непосредственно из рисунка получаем:

$$a_1 = r_2 \bar{i}_2 + G r_1 i_2;$$

$$a_2 = G r_1 \bar{i}_2.$$

Операция подавления записи ($i_1 = 1$). Как и операция подавления чтения, операция подавления записи осуществляется путем запрета операции записи ($a_1 = 0$) и введения вместо нее операции чтения ($a_2 = Gr_2$), как показано на рис. 7.16.

Непосредственно из рисунка получаем:

$$a_1 = r_2\bar{i}_1;$$
$$a_2 = Gr_1\bar{i}_1 + Gr_2 i_1.$$

$i_1 i_2$ 00	01	11	10
$a_1 = r_2$ $a_2 = Gr_1$	$a_1 = Gr_1$ $a_2 = 0$	$a_1 = 0$ $a_2 = 0$	$a_1 = 0$ $a_2 = Gr_2$

Рис. 7.17

Интерфейсные уравнения. Сигналы запроса для всех комбинаций сигналов подавления чтения и записи показаны на рис. 7.17. Непосредственно из этого рисунка имеем:

$$a_1 = \bar{i}_1\bar{i}_2 r_2 + \bar{i}_1 i_2 Gr_1; \tag{7.3}$$
$$a_2 = i_1\bar{i}_2 Gr_1 + i_1\bar{i}_2 Gr_2. \tag{7.4}$$

Эти уравнения мы будем называть интерфейсными уравнениями.

7.4. ЗАДАЧИ И РЕШЕНИЯ

В данном параграфе демонстрируются изложенные принципы проектирования на задачах и их полных решениях. Внимание читателя обращается на тот факт, что, хотя для реализации постав-

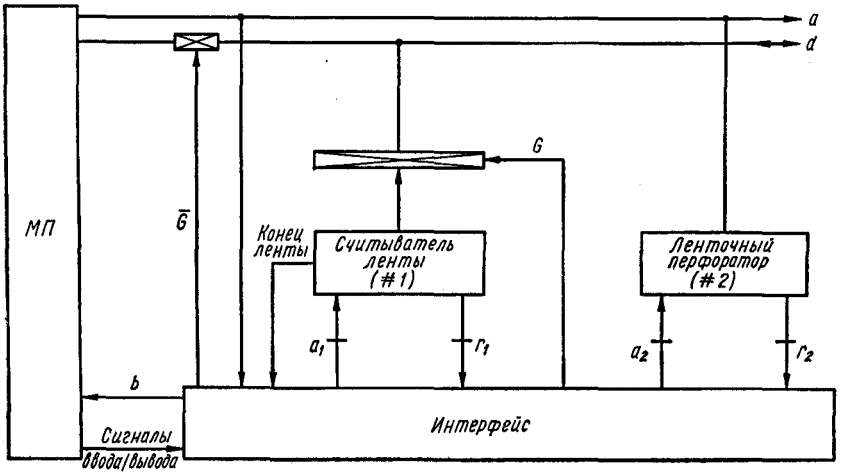


Рис. 7.18

ленных задач используется МП *In8080*, описываемые процедуры применимы для всех типов микропроцессоров. Особо необходимо отметить, что первые три шага алгоритма проектирования выполняются вообще без ссылки на тип микропроцессора.

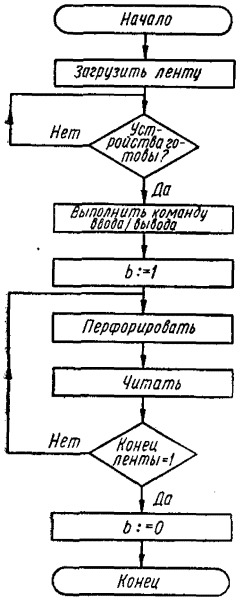


Рис. 7.19

Задача 1. Копирование ленты. Скопировать данную перфоленту, используя режим ППД.

Решение.

Шаг 1 — формирование технического задания. Целью проектирования является разработка устройства, пересылающего данные от источника к приемнику в режиме ППД. Никакая обработка данных не требуется.

Шаг 2 — внешнее проектирование. Считыватель и перфоратор являются устройствами типа запрос/ответ. Кроме того, считыватель генерирует сигнал конца ленты *(e. o. t)*.

Шаг 3 — системное проектирование. Структурная схема проектного решения показана на рис. 7.18. Алгоритм ее функционирования представлен в виде схемы алгоритма на рис. 7.19.

Шаг 4 — проектирование аппаратурной части. Из рис. 7.18 видно, что интерфейс должен формировать сигналы a_1, a_2, b и G.

Сигналы a_1 и a_2 могут быть получены непосредственно из интерфейсных уравнений (7.3) и (7.4) подстановкой 0 вместо i_1 и i_2:

$$a_1 = r_2;$$
$$a_2 = Gr_1.$$

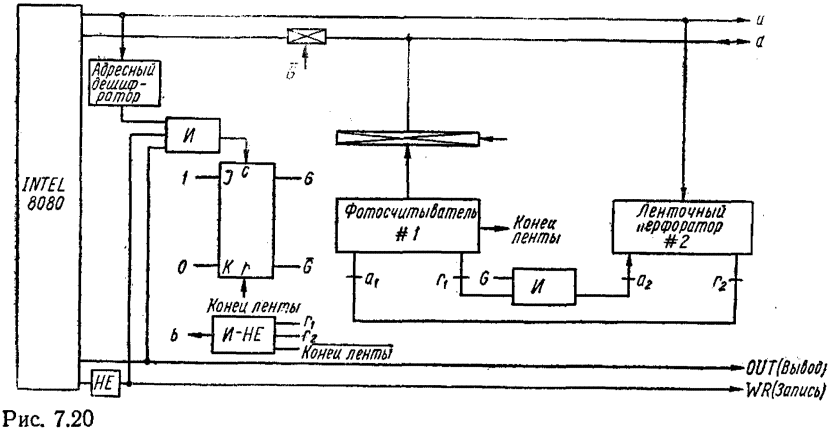


Рис. 7.20

Выражение для сигнала b следующее (рис. 7.20):

$$b = \bar{r}_1 + \bar{r}_2 + e.o.t.$$

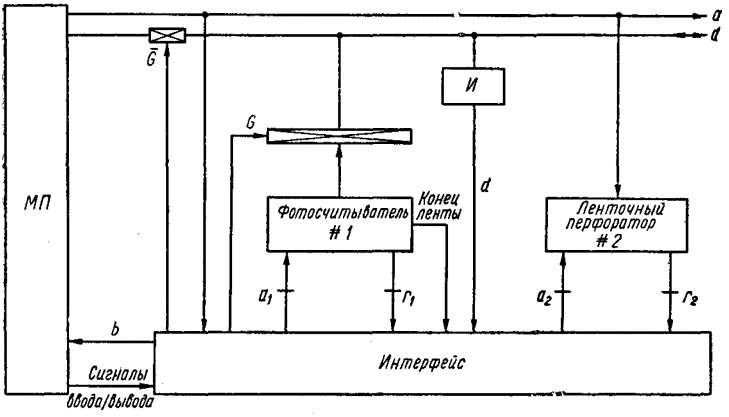


Рис. 7.21

Для формирования сигнала G (управление работой/остановом) используется JK-триггер (рис. 7.20). Установка этого триггера производится командой ввода/вывода. Сбрасывается триггер сигналом *«Конец ленты»* (*e. o. t*).

Шаг 5 — проектирование программного обеспечения. За исключением команды ввода/вывода, запускающей обмен данными по каналу ППД, никакое другое программное обеспечение не требуется.

Задача 2. Очистка ленты. Скопировать перфоленту, удалив все забитые символы (т. е. символы, состоящие из единиц).

Решение.

Шаг 1 — формирование технического задания. Целью проектирования является выборочное воспроизведение принимаемых данных. В нашем случае забитые символы не подлежат воспроизведению.

Шаг 2 — внешнее проектирование. Фотосчитыватель с ленты и ленточный перфоратор являются устройствами типа запрос/ответ. Кроме того, фотосчитыватель генерирует сигнал *«Конец ленты»* (*e. o. t*).

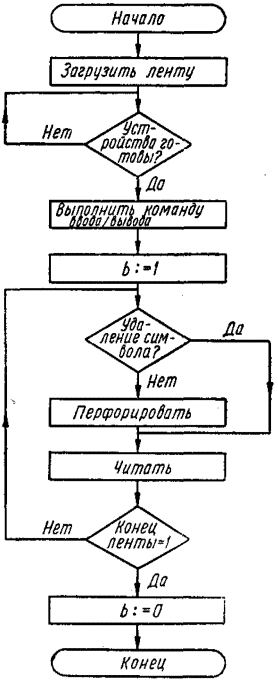


Рис. 7.22

Шаг 3 — системное проектирование. Структурная схема разрабатываемого устройства показана на рис. 7.21, а схема алгоритма его функционирования — на рис. 7.22.

Шаг 4 — проектирование аппаратурной части. Из рис. 7.21 видно, что разрабатываемый интерфейс должен формировать сигналы a_1, a_2, b и G.

Сигналы a_1 и a_2 получаются непосредственно из интерфейсных уравнений (7.3) и (7.4) подстановкой d и 0 вместо i_1 и i_2

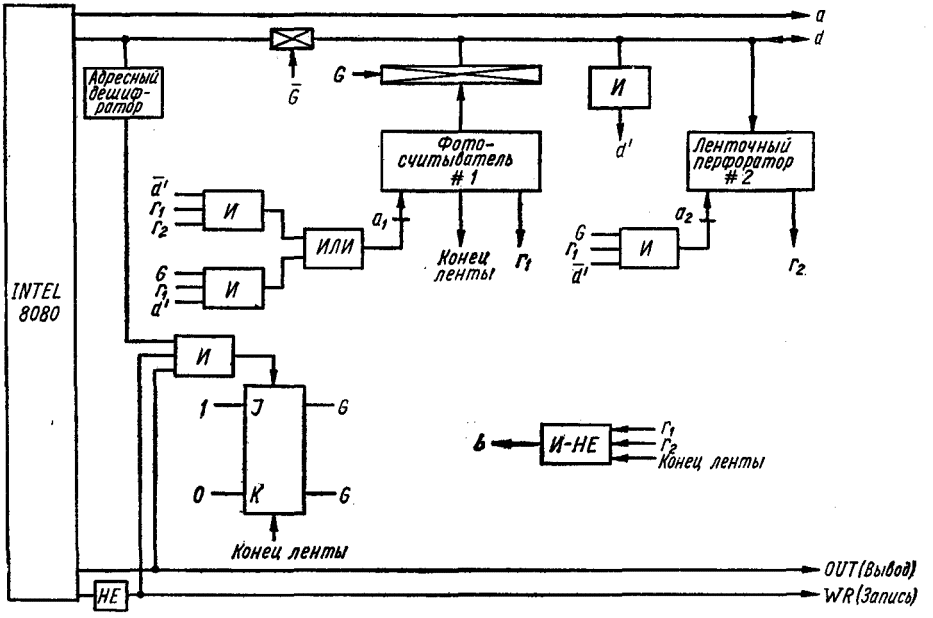


Рис. 7.23

соответственно. Сигнал d формируется путем объединения по схеме *И* всех восьми разрядов шины данных, т. е. состояние $d=1$ служит признаком наличия забитого символа. Сигнал d анализируется только в моменты, когда $r_1=1$, т. е. в то время, когда данные, выдаваемые фотосчитывателем, стабильны. Поэтому

$$a_1 = \bar{d} r_1 r_2 + G r_1 d;$$

$$a_2 = \bar{d} G r_1.$$

Уравнение для сигнала b имеет вид:

$$b = \bar{r}_1 + \bar{r}_2 + e.o.t.$$

Сигнал G, как и в задаче 1, формируется JK-триггером (рис. 7.23). Этот триггер устанавливается пусковой командой

ввода/вывода, а сбрасывается автоматически сигналом *«Конец ленты»*.

Шаг 5 — проектирование программного обеспечения. Разработанное устройство не требует никакого программного обеспечения, кроме команд ввода/вывода, запускающих передачу данных.

7.5. СПИСОК ЛИТЕРАТУРЫ

1. Zissos D. Problems and Solutions in Logic Design. Oxford University Press, 1976.

2. Zissos D., Duncan F. G., Colin T. J. Logic-free Data Channels, — Electronics Letters, vol. 10, № 17, August, 1974.

УСТРОЙСТВА ТИПА ЗАПРОС/ОТВЕТ

В данном приложении вводится определение устройств типа запрос/ответ и объясняется методика их реализации.

П1.1. УСТРОЙСТВО ЗАПРОС/ОТВЕТ

В 1974 г. в [1, 2] было показано, что интерфейс между двумя устройствами типа запрос/ответ состоит из двух проводников. Устройства типа запрос/ответ имеют два вывода: вывод запроса и вывод ответа (рис. П1.1). Сигналы a и r несут следующую информацию.

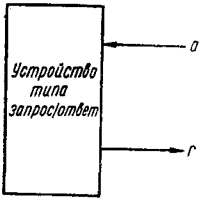


Рис. П1.1

Сигнал a. Переход от «0» до «1» этого сигнала запускает (активизирует) устройство. Запуск невозможен, когда $r=0$.

Сигнал r. Этим ответным сигналом определяется доступность ($r=1$) или недоступность ($r=0$) устройства.

В настоящее время большинство устройств управляются несколькими сигналами и не могут быть смоделированы приведенной моделью типа запрос/ответ. Такие устройства, однако, могут быть легко приведены к виду запрос/ответ с помощью логической схемы, чувствительной к фронтам.

П1.2. ЛОГИЧЕСКАЯ СХЕМА, ЧУВСТВИТЕЛЬНАЯ К ФРОНТАМ

Схема, чувствительная к фронтам, показана на рис. П1.2. Схема формирует сигналы состояния устройства и генерирует правильные последовательности управляющих сигналов для запусков устройства при изменении уровня сигналов запуска от «0» к «1». Кроме того, она генерирует сигнал ответа r. Реализация такой логической схемы довольно проста, если применить методику, описанную в гл. 1.

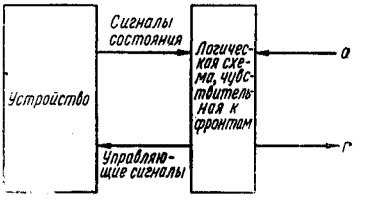


Рис. П1.2

Основная трудность, с которой приходится сталкиваться пользователю, заключается в правильной интерпретации сигналов управления и состояния устройства по предоставляемой изготовителем документации. Рассмотрим пример несколько тенденциозного содержания, заключающийся в том, что некоторый пользователь достаточно хорошо знаком с семантикой сигналов состояния и управления, однако не хочет использовать устройство в том виде, в каком оно поставляется изготовителем. Ниже дано несколько общих рекомендаций по проектированию логических схем, чувствительных к фронтам.

В связи с тем, что требования к входным сигналам для всех устройств, снабженных логической схемой, чувствительной к фронту, одинаковы, сигнал запроса не используется непосредственно для запуска устройства, а исполь-

зуется для перевода схемы в следующее по диаграмме состояние. В этом новом состоянии генерируется первый из сигналов управления и обрабатываются соответствующие ответные сигналы состояния. Когда же устройство закончило выполнение действий, вызванных поступившими управляющими сигналами, логическая схема, чувствительная к фронту, генерирует следующий управляющий сигнал. Процесс продолжается до тех пор, пока устройство не выполнит действия, «заказанные» последним управляющим сигналом. Тогда схема возвращается в свое исходное состояние для $a=0$. Благодаря этому обеспечивается чувствительность логической схемы только к фронту сигнала запроса и допускается автономный режим работы при подсоединении его выхода готовности (состояния) к входу запроса. Рассмотрим разработку логической схемы, чувствительной к фронту, на примере цифропечатающего устройства.

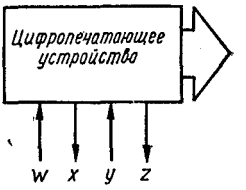


Рис. П1.3

Пример 16. Спроектировать логическую схему, чувствительную к фронту для цифропечатающего устройства (рис. П1.3), которое имеет следующие характеристики ввода/вывода.

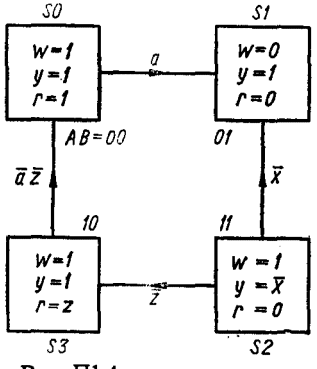


Рис. П1.4

Контакт w. Подача уровня «0» на этот контакт устанавливает печатающий узел в соответствие с вводимыми данными.

Контакт x. Пока печатающий узел устанавливается, $x=0$. Сигнал принимает значение 1, когда печатающий узел окончательно установится.

Контакт y. Подача нулевого уровня на этот контакт вызывает срабатывание молоточков печатающего узла и последующую протяжку бумаги на одну позицию.

Контакт z. Сигнал $z=0$ в течение срабатывания молоточков и протяжки бумаги, остальное время $z=1$.

В данном примере сигналы a, x и z являются входными сигналами логической схемы, а w, y и r — выходными. Диаграмма состояний разрабатываемой схемы показана на рис. П1.4. Непосредственно из диаграммы получаем:

условие установки $A = B\bar{x}$;
условие сброса $A = \bar{B}\bar{a}z \rightarrow$ инверсия $\rightarrow B + a + \bar{z}$;
условие установки $B = \bar{A}a$;
условие сброса $B = \bar{A}\bar{z} \rightarrow$ инверсия $\rightarrow \bar{A} + z$.

Таким образом, уравнения схемы имеют вид:

$$A = B\bar{x} + A(B + a + \bar{z});$$
$$B = \bar{A}a + B(\bar{A} + z);$$
$$w = \overline{S_1} = \overline{A}\overline{B} = A + \overline{B};$$
$$y = S_0 + S_1 + S_2\bar{x} + S_3$$
$$= \overline{S_2} + S_2\bar{x}$$
$$= \overline{S_2} + \bar{x}$$
$$= \overline{A} + \overline{B} + \bar{x}.$$

Схема, соответствующая реализации этих уравнений, показана на рис. П1.5. Данная схема соответствует схеме, чувствительной к фронту, и используется в цифропечатающих устройствах.

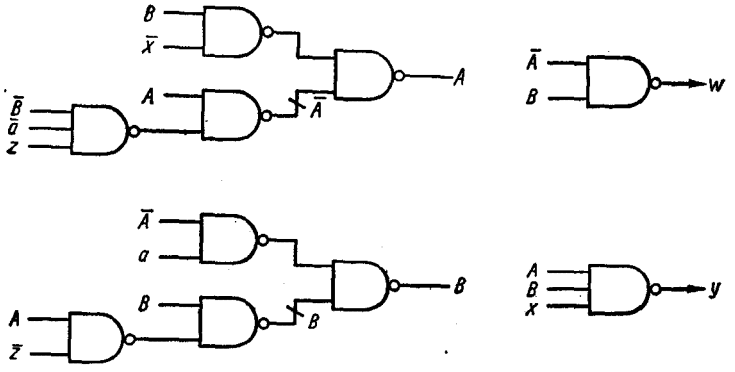


Рис. П1.5

П1.3. СПИСОК ЛИТЕРАТУРЫ

1. **Zissos D., Duncan F. G., Collin T. J.** 'Logic-free Data Channels;— Electronics Letters. Vol. 10, № 17, August, 1974.

2. **Collin T. J.** 'Logic-free Data Channels; M. Sc. Thesis, University of Calgary, 1974.

Приложение 2

МИКРОПРОЦЕССОР *Intel 8085*

П2.1. ОБЩИЕ ПОЛОЖЕНИЯ

Микропроцессор *In 8085* является эволюционным преемником микропроцессора *In 8080**. Его выпуск стал возможен благодаря усовершенствованию технологии за последние 10 лет, а также благодаря приобретению опыта в разработке и использовании микропроцессорных систем. Основные отличительные черты этого МП, с точки зрения системного разработчика, следующие.

1. Одно напряжение питания (+5В). Эта особенность полезна для портативной аппаратуры, она позволяет снизить общую стоимость разрабатываемой системы.

2. Наличие однофазной синхронизации. Использование однофазной синхронизации значительно ослабляет временные ограничения на интерфейсные сигналы. Минимальная и максимальная частоты синхронизации соответственно 0,5 и 3 МГц. Вся система синхронизации встроена в кристалл, и для функционирования необходим только внешний кварцевый генератор или *RC*-генератор. Сигнал синхронизации *Ф* выводится на контакт *37* (рис. П2.1), все внутренние переходы осуществляются по спаду синхросигнала *Ф*.

Необходимо обратить внимание на тот факт, что требуемые для функционирования МП две неперекрывающиеся по фазе последовательности синхросигналов генерируются внутри кристалла от внешнего источника. Одна из этих последовательностей выдается потребителю.

3. Уменьшенный набор семейства. Повышение уровня интеграции позволило реализовать минимальную архитектуру микро-ЭВМ на трех кристаллах: 8085 (микропроцессор), 8155 (ОЗУ) и 8355/8755 (ПЗУ/ППЗУ).

---

* Более подробная информация по микропроцессору *In 8085* приведена в фирменном руководстве [1].

$\longrightarrow a^*$

$\longrightarrow d^*$

(35) $\overline{READY}$ ← Подача «0» на этот вход переводит МП в состояние ожидания, а подача «1» выводит его из этого состояния. Шина стробируется в течение машинных тактов $T2$ и T_w каждого цикла по фронту фазы $\Phi \rightarrow$ (рис. П2.5)

(6) $TRAP$ ← Немаскируемое прерывание. Переход 0 → 1 вызывает передачу управления по адресу $000\,044_8$

(9) $\overline{RST\ 7,5}$ ← Переход 0 → 1 вызывает передачу управления по адресу $000\,075_8$

(8) $\overline{RST\ 6,5}$ ← Подача «1» вызывает передачу управления по адресу $000\,065_8$

(7) $\overline{RST\ 5,5}$ ← Подача «1» вызывает передачу управления по адресу $000\,055_8$

(10) *INTEL 8085* $INTR$ ← Подача «1» прерывает программу (асинхронно). Команда *RST* или команда *CALL* выдается на шины в течение цикла обработки прерывания

(39) $HOLD$ ← Подача «1» переводит *a*, *d*, $\overline{RD}$, $\overline{WR}$, $I/\overline{OM}$, и *ALE* в третье состояние

(36) $RESET\ IN$ ← Подача «0» сбрасывает программный счетчик и переводит МП в состояние $M1 \cdot T1$ (рис. П2.5), устанавливается маска *RST* и запрещается сигнал *IN TR*

(29) S_0 →

(33) S_1 →

Информация о состоянии →

S_0	S_1	
0	0	Программный останов
0	1	Запись
1	0	Чтение
1	1	Извлечение команды

(34) $I/\overline{OM}^*$ → «1» индицирует выполнение ввода/вывода

(31) $\overline{WR}^*$ → «0» индицирует вывод данных

(32) $\overline{RD}^*$ → «0» индицирует режим ввода по шине данных

} Не только для устройств ввода/вывода

(11) $\overline{INTA}$ → Этот сигнал генерируется в течение следующего командного цикла после приема сигнала *IN TR* (запрос прерывания). Он используется вместо сигнала $\overline{RD}$ и имеет те же временные характеристики

(38) $HLDA$ → «1» подтверждает прием захвата *HOLD*. Шины будут переведены в третье состояние в начале следующего синхроцикла

(30) ALE^* → Спад этого сигнала можно использовать для стробирования младших разрядов адресной шины

(3) $RESET\ OUT$ → Индицирует сброс микропроцессора

* Находятся в третьем состоянии во время программного останова.

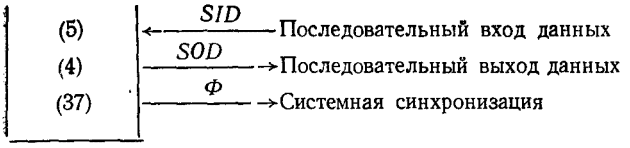


Рис. П2.1

4. Сигналы микропроцессора приведены на рис. П2.1. В общем, они спроектированы весьма разумно и достаточно четко определены, хотя имеется два исключения: а) контакт $I/\overline{OM}$ (контакт *34* рис. П2.1) переводится в третье состояние в момент выполнения команды *«Останов»*, однако, если в данной системе используется нагрузочный резистор, то сигнал $I/\overline{OM}$ принимает уровень логической «1» при останове микропроцессора, ошибочно индицируя системе, что выполняется команда ввода/вывода; б) в микропроцессоре не предусмот-

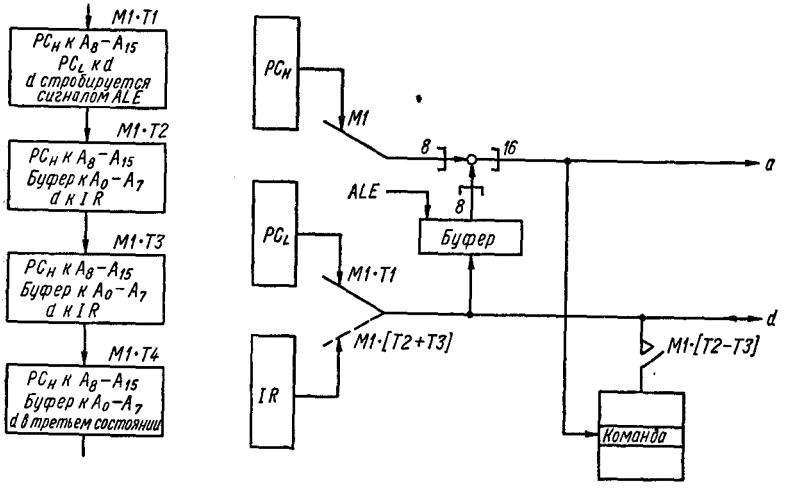


Рис. П2.2

рен сигнал ожидания, как это сделано в *In 8080,* который бы индицировал, что прибор находится в состоянии ожидания.

5. Мультиплексируемая шина данных. Микропроцессор *In 8085* использует мультиплексирование шины данных. Процесс мультиплексирования лучше понять при рассмотрении рис. П2.2 и П2.3. Из этих рисунков видно, что в течение такта *T1* каждого машинного цикла шина данных *d* не несет информации. Из этого следует, что для 8-битовой машины с 16-битовым полем адреса 8 из 16-адресных сигналов можно ввести на шину данных в течение такта *T1* и простробировать до того, как шина данных будет использована для обмена с памятью или ввода/вывода. Таким путем можно освободить 8 выводов, которые можно применить для других целей. Именно такой метод мультиплексирования данных и применен в микропроцессоре *In 8085.* На рис. П2.2 показано мультиплексирование шин данных в цикле выборки команды, обозначенном как *M1. Сигнал ALE* (*Adress Latch Enable*, строб выдачи адреса) пред-

ставляет собой импульс, генерируемый микропроцессором *In 8085* в каждом машинном цикле перед переходом в состояние *T2*. Спад сигнала *ALE* можно использовать для стробирования выдаваемой по шинам данных адресной информации. Временные диаграммы описываемых процессов приведены в работе [1].

6. Набор команд. Под набором команд подразумевается набор операций, которые может выполнять микропроцессор. Микропроцессор *In 8085* имеет такой же набор команд, как микропроцессор *In 8080*, за исключением двух дополнительных команд — *SIM* (*Set Interrupt Masks — «Установить маску прерывания»*) и *RIM* (*Read Interrupt Masks — «Читать маску прерывания»*). Команда *SIM* пересылает содержимое аккумулятора в регистр маски прерывания,

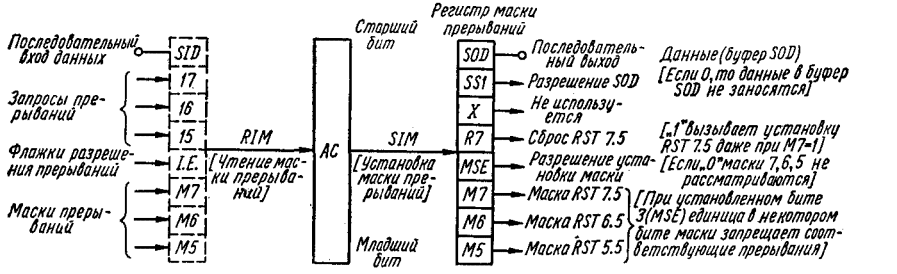


Рис. П2.4

а команда *RIM* позволяет прочесть в аккумулятор информацию о состоянии (рис. П2.4).

7. Режим ожидания/счета. Проектирование и реализация систем типа ожидания/счета на микропроцессоре *In 8085*, как и на любом другом, не представляет затруднений. Микропроцессор *In 8085* в качестве центрального устройства ожидания/счета рассматривается в следующем параграфе. Проектирование логической схемы ожидания/счета было бы несколько проще, если бы пользователю был доступен сигнал ожидания. Сигнал ожидания, как уже указывалось в гл. 3, индицирует, что микропроцессор находится в состоянии ожидания некоторого сигнала.

8. Режим проверки и пропуска. Последовательность разработки систем, работающих в режиме проверки и пропуска, описывается в параграфе П2.3.

9. Режим прерывания. Возможна обработка следующих прерываний:

а) немаскируемое прямое векторное прерывание, *TRAP*, на контакт *6* (рис. П2.1);

б) три прямых векторных прерывания на контакты *7, 8* и *9;*

в) восемь косвенных векторных прерываний, как и для микропроцессора *In 8080.*

10. Режим ПДП. Никаких особенностей для данного микропроцессора при применении его в режиме ПДП нет.

11. Режим ППД. Как и для режима ПДП, в данном случае никаких особенностей для микропроцессора *In 8085* нет.

П2.2. СИСТЕМЫ ОЖИДАНИЯ/СЧЕТА

Как и все выпускаемые в настоящее время микропроцессоры МП *In 8085* не выполняет циклов ожидания/счета. Как уже объяснялось в гл. 3, под такими циклами понимаются циклы ввода/вывода по адресам ожидания/счета A_D в течение которых микропроцессор автоматически входит в состояние ожидания, которое он покидает, когда сигнал на шине счета изменяется от «0» до «1». Пользователю предоставляется решение задачи разработки логической схемы ожидания/счета, позволяющей инициировать цикл ожидания/счета. Структурная схема изображена на рис. 3.12.

		0							
		$D_2D_1D_0$							
		000	010	110	100	101	111	011	001
	000	*NOP* 000	$M_p := A$ 002	006	004	005	Сдвиг 007	$P := P+1$ 003	$P := II$ 001
	010		022	026	024	025	027	023	021
	110	*SIM*	$M1 := A$ 062	$r := I$ 066	$r := r+1$ 064	$r := r-1$ 065	$c := 1$ 067	063	061
	100	*RIM*	$M1 := HL$ 042	046	044	045	*BCD* 047	043	041
	$D_5D_4D_3$ 101		$HL := M1$ 052	056	054	055	$A := \overline{A}$ 057	053	051
0	111		$A := M1$ 072	076	074	075	$c := \overline{c}$ 077	$P := P-1$ 073	071
	011		$A := M_p$ 032	036	034	035	Сдвиг 037	033	$HL := HL+P$ 031
D_7	001		012	016	014	015	017	013	011
	$+c$ 001	210	212	216	214	215	217	213	211
	$-c$ 011	230	232	236	234	235	237	233	231
	(—) 111	270	272	276	274	275	277	273	271
	$\forall$ 101	250	252	256	254	255	257	253	251
1	$D_5D_4D_3$ $\wedge$ 100	240	242	246	$A := A\ op\ r$ 244	245	247	243	241
	$\vee$ 110	260	262	266	264	265	267	263	261
	— 010	220	222	226	224	225	227	223	221
	+ 000	200	202	206	204	205	207	203	201

Регистр r		Условие		Оператор op	
000	B	$\overline{z}$	не нуль	+	
010	D	$\overline{c}$	не перенос	—	
110	Mhl	$\overline{n}$	не меньше нуля	$\vee$	
100	H	$\overline{p}$	нечет	$\wedge$	
101	L	p	чет	$\forall$	
111	A	n	меньше нуля	сравнение ($z := A = r$,	$c := A < r$)
011	E	c	перенос	$-c$ ($A := A - r - c$)	
001	C	z	нуль	$+c$ ($A := A + r + c$)	

Рис. П2.3. Карта команд *INTEL 8085*

D_6	1							
	$D_5D_4D_3$							
	001	011	111	101	100	110	010	000	
	101	103	107	105	104	106	102	100	*B*
			r1 := r2			*r1 := Mhl*	*r1 := r2*		
	121	123	127	125	124	126	122	120	*D*
	161	163	167	165	164	Останов 166	162	160	
	141	143	147	145	144	146	142	140	*H*
	151	153	157	155	154	156	152	150	*L*
	171	173	*r1 := r2* 177	175	174	*r1:=Mhl* 176	*r1 := r2* 172	170	*A*
	131	133	137	135	134	136	132	130	*E*
	111	113	117	115	114	116	112	110	*C*
	RET 311		317	*CALL* 315	314	316	312	310	*z*
		IN 333	337		334	336	332	330	*c*
	SP:=HL 371	*EI* 373	*REST ART* 377		Условный 374	376	Условный 372	Условный 370	*n*
	PC:=HL 351	353	357		вызов 354	356	переход 352	возврат 350	*p*
	341	343	347	345	344	*A:=A op 1* 346	342	340	$\overline{p}$
	361	*DI* 363	367	365	364	366	362	360	$\overline{n}$
	POP p 311	*OUT* 323	327	*PUSH p* 325	324	326	322	320	$\overline{c}$
	301	*JMP* 303	307	305	304	306	302	300	$\overline{z}$

Пара регистров *p*

00 *BC*
01 *DE*
11 *SP**
10 *HL*

Сдвиг (*A*)

Логический влево
Логический вправо
Арифметический вправо
Арифметический влево

* *AF* для команд *POP, PUSH*

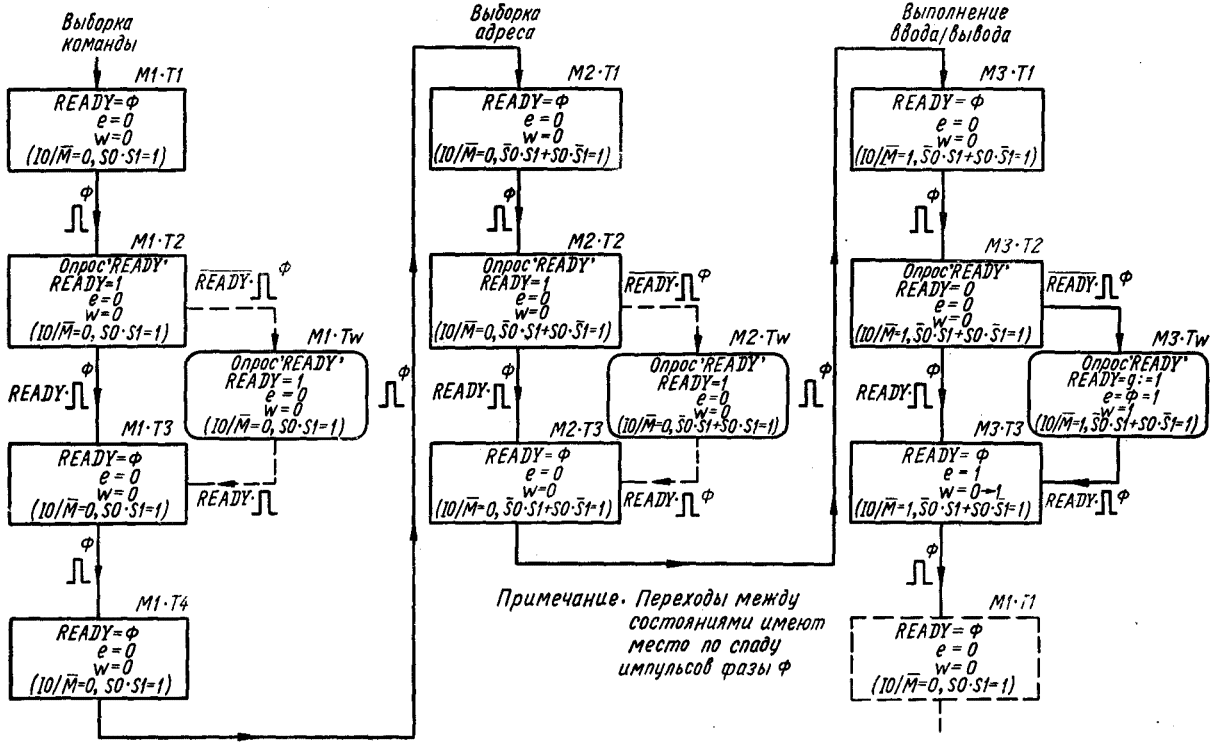

Выборка команды
M1·T1
READY=Ф
e=0
w=0
(IO/M=0, S0·S1=1)
Ф
M1·T2
Опрос 'READY'
READY=1
e=0
w=0
(IO/M=0, S0·S1=1)
READY·Ф
M1·Tw
Опрос 'READY'
READY=1
e=0
w=0
(IO/M=0, S0·S1=1)
READY·Ф
M1·T3
READY=Ф
e=0
w=0
(IO/M=0, S0·S1=1)
READY·Ф
M1·T4
READY=Ф
e=0
w=0
(IO/M=0, S0·S1=1)
Выборка адреса
M2·T1
READY=Ф
e=0
w=0
(IO/M=0, S0·S1+S0·S1=1)
M2·T2
Опрос 'READY'
READY=1
e=0
w=0
(IO/M=0, S0·S1+S0·S1=1)
M2·Tw
Опрос 'READY'
READY=1
e=0
w=0
(IO/M=0, S0·S1+S0·S1=1)
M2·T3
READY=Ф
e=0
w=0
(IO/M=0, S0·S1+S0·S1=1)
Выполнение ввода/вывода
M3·T1
READY=Ф
e=0
w=0
(IO/M=1, S0·S1+S0·S1=1)
M3·T2
Опрос 'READY'
READY=0
e=0
w=0
(IO/M=1, S0·S1+S0·S1=1)
M3·Tw
Опрос 'READY'
READY=q:=1
e=Ф=1
w=1
(IO/M=1, S0·S1+S0·S1=1)
M3·T3
READY=Ф
e=1
w=0→1
(IO/M=1, S0·S1+S0·S1=1)
M1·T1
READY=Ф
e=0
w=0
(IO/M=0, S0·S1=1)
Примечание. Переходы между состояниями имеют место по спаду импульсов фазы Ф

Рис. П2.5

Рассмотрим диаграмму внутренних состояний микропроцессора *In 8085* во время выполнения цикла ввода/вывода (рис. П2.5). Как и для микропроцессора *In 8080*, для реализации цикла ожидания/счета используется переход микропроцессора из состояния *M3·T2* в состояние $M3.T_w$ (вместо перехода в состояние *M3·T3*). Переход в состояние *M3.T3* инициируется переходом сигнала из «0» в «1» на шине счета *g* (рис. П2.6).

В соответствии с рис. П2.5 микропроцессор *In 8085* переходит в состояние ожидания благодаря подаче нулевого уровня на линию *READY* (контакт *35*). Так как данная линия опрашивается микропроцессором в течение состояний *T2* и T_w, логический «0» на нее необходимо подать, когда микропроцессор *In 8085* переходит в состояние *M3·T2*, а на шинах поддерживается адрес ожидания/счета A_w. Нулевой уровень на линии *READY* должен быть до тех пор, пока сигнал счета *g* не изменится с «0» на «1», при этом необходимо подать на линию *READY* единичный уровень. Это позволит микропроцессору перейти в состояние *M3·T3* и продолжить свое нормальное функционирование. На рис. П2.1 показано, что в течение выполнения цикла команды ввода/вывода на контакте *34* поддерживается уровень логической «1». Этот сигнал обозначается как $I/\overline{OM}$. Уровень логической «1» на этом контакте также может быть получен при отработке команды останова, как это указывалось в параграфе П2.1 (пункт 4). Очевидно, что логическая схема ожидания/счета не должна опрашивать контакт $I/\overline{OM}$ во время программного останова процессора. Из рис. П2.1 видно, что в течение программного останова оба сигнала со-

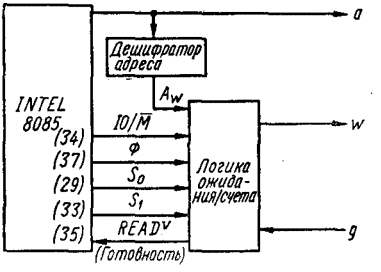


Рис. П2.6

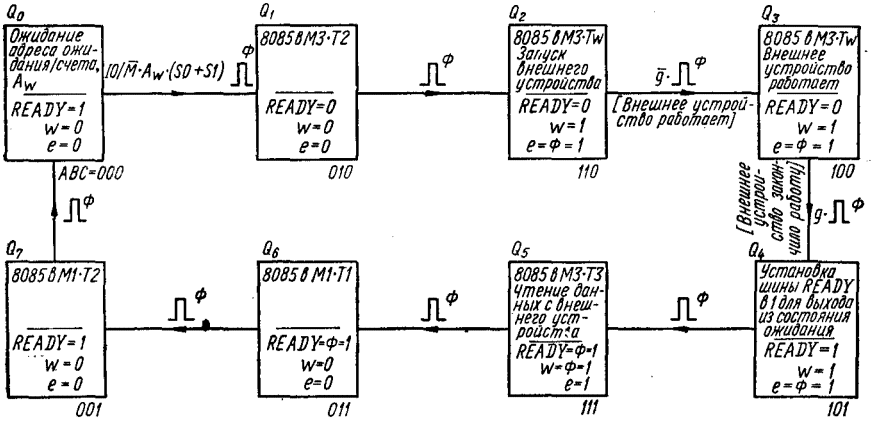


Рис. П2.7

стояния S_0 и S_1 равны 0. Таким образом, используя вентиль *И* для сигналов $I/\overline{OM}$ и $S_0 \cdot S_1$ получим искомый сигнал логической схемы ожидания/счета $I/\overline{OM}$ $(S_0 + S_1)$.

Итак, кроме сигналов $I/\overline{OM}$, A_w и *Ф* логическая схема ожидания/счета должна анализировать сигналы состояния S_0 и S_1. Для упрощения мультиплек-

сирование и демультиплексирование на шинах w и g не показано. Интересующийся читатель может найти подробности на рис. 3.10.

Соответствующая диаграмма внутренних состояний логической схемы ожидания/счета показана на рис. П2.7,*а*. Функционирует она следующим образом.

Нормальное состояние схемы Q_0, схема находится в этом состоянии, пока *In 8085* активно функционирует, а периферийные устройства, работающие в режиме ожидания/счета, неактивны. *In 8085* поддерживается в активном состоянии благодаря наличию на шине *READY* высокого уровня. В этом состоянии схема ожидает появления сигнала адреса ожидания/счета A_w. С этой целью опрашивается адресная шина микропроцессора, пока он находится в состоянии *M3·T1* (рис. П2.5). Когда *In 8085* находится в состоянии *M3·T1*, то

$$I/O\overline{M} = 1;\ S_0 + S_1 = 1.$$

Таким образом, можно использовать сигнал $I/O\overline{M}(S_0+S_1)A_w$ для перехода в следующее состояние. Так как переход в схеме и в микропроцессоре про-

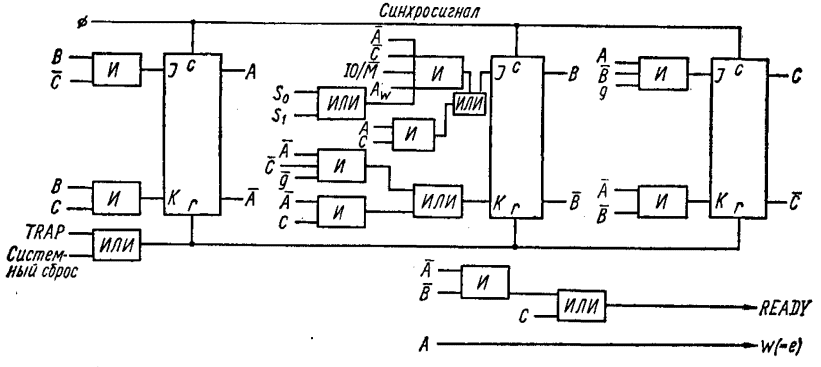


Рис. П2.8

исходит по спаду импульсов фазы Φ, схема переходит в состояние Q_1 в то же самое время, когда микропроцессор переходит в состояние *M3·T2* (рис. П2.5). В состоянии Q_1 на линию *READY* подается низкий уровень. Благодаря этому микропроцессор переводится в состояние ожидания по следующему синхроимпульсу. Одновременно схема ожидания/счета переходит в состояние Q_2, при этом формируется сигнал ожидания w. Изменение сигнала w от «0» к «1» используется для запуска внешних устройств. Для того чтобы быть уверенными, что связь с устройством не оборвалась, схема не переводится в состояние Q_3 до тех пор, пока сигнал g не станет равным 0. Последнее обстоятельство указывает на то, что внешнее устройство начало отвечать на запрос. Состояние Q_3 сохраняется до тех пор, пока устройство не ответит на запрос, т. е. пока сигнал g не станет вновь равным 1. В состоянии Q_4 на линию *READY* подается высокий уровень, позволяя следующему синхроимпульсу вывести микропроцессор из состояния ожидания. Одновременно логическая схема ожидания/счета переходит в состояние Q_5. Данные считываются в аккумулятор, если внешнее устройство является источником, когда микропроцессор находится в состоянии *M3·T3*. Это значит, что в состоянии Q_5 сигнал разрешения e должен быть равным 1.

Из состояния Q_5 можно непосредственно возвратиться в состояние Q_0. Однако в таком случае диаграмма будет иметь шесть рабочих состояний и два — неиспользуемые. Во избежание такой ситуации по причинам, описанным в параграфе 1.6, введем дополнительные состояния Q_6 и Q_7 для нормальной работы схемы.

Очевидно, что сигнал *READY* анализируется только в состоянии Q_0, когда микропроцессор находится в состояниях *M3·T2* или *M3·T*$_w$. Это связано с тем,

что линия *READY* опрашивается микропроцессором только в состояниях *T2* и T_w (рис. П2.1).

Так как *In 8085* входит в состояние ожидания в течение цикла ввода/вывода, сигнал e может быть приравнен сигналу w (см. уравнение (3а)).

Чтобы исключить нежелательные броски сигналов на линии ожидания w, воспользуемся свободным от состязания кодированием для определения требуемых восьми состояний. Примеры таких кодов приведены на рис. П1.8, а конкретное кодирование для рассматриваемого случая — на рис. П2.7. Непосредственно из этого рисунка получаем уравнения:

$$S_A = Q_1 = \overline{A}B\overline{C}, \qquad \text{поэтому } J_A = B\overline{C};$$

$$R_A = Q_5 = ABC, \qquad \text{поэтому } K_A = BC;$$

$$S_B = Q_0 I/\overline{OM}\,(S_0 + S_1)\,A_w + Q_4 = \overline{A}\overline{B}\overline{C}I/\overline{OM}\,(S_0 + S_1)\,A_w + A\overline{B}\overline{C}, \qquad \text{поэтому } J_B = \overline{A}\overline{C}I\overline{OM}\,(S_0 + S_1)\,A_w + AC;$$

$$R_B = Q_2\overline{g} + Q_6 = A\overline{B}\overline{C}\overline{g} + \overline{A}BC, \qquad \text{поэтому } K_B = \overline{A}\overline{C}\overline{g} + \overline{A}C;$$

$$S_C = Q_3 g = A\overline{B}\overline{C}g, \qquad \text{поэтому } J_C = A\overline{B}g;$$

$$R_C = Q_7 = \overline{A}BC, \qquad \text{поэтому } K_C = \overline{A}\overline{B};$$

$$READY = Q_0 + Q_4 + Q_7 + (Q_5) + (Q_6) = \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + \overline{A}BC + (ABC) + (\overline{A}BC) = \overline{A}\overline{B} + C;$$

$$e = w = Q_2 + Q_3 + Q_4 + (Q_5) = A\overline{B}\overline{C} + A\overline{B}\overline{C} + A\overline{B}\overline{C} + (ABC) = A;$$

Сигнал сброса = Системный сброс + *TRAP*.

Системный сброс позволяет осуществить синхронизацию в системе, а сигнал *TRAP* позволяет ей покинуть состояние ожидания в особых случаях. Соответствующая схема показана на рис. П2.8.

П2.3. СИСТЕМЫ ПРОВЕРКИ И ПРОПУСКА

Структурная схема микропроцессорной системы проверки и пропуска для одного внешнего устройства показана на рис. 4.2, а алгоритм ее функционирования — на рис. 4.1.

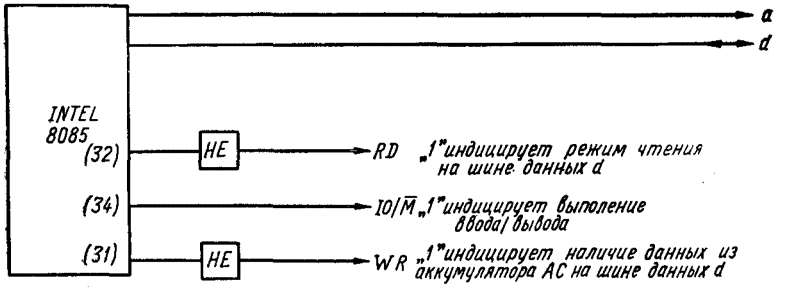


Рис. П2.9

Сигналы ввода/вывода, как и во всех микропроцессорах, генерируются во время выполнения команд ввода/вывода. Для микропроцессора *In 8085* все пересылки ввода/вывода осуществляются через аккумулятор. Непосредственно

может адресоваться до 256 вводных и 256 выходных устройств. Выполнение команд ввода/вывода не изменяет состояния флажков микропроцессора.

Сигналы ввода/вывода показаны на рис. П2.9. Соответствующие им вре-

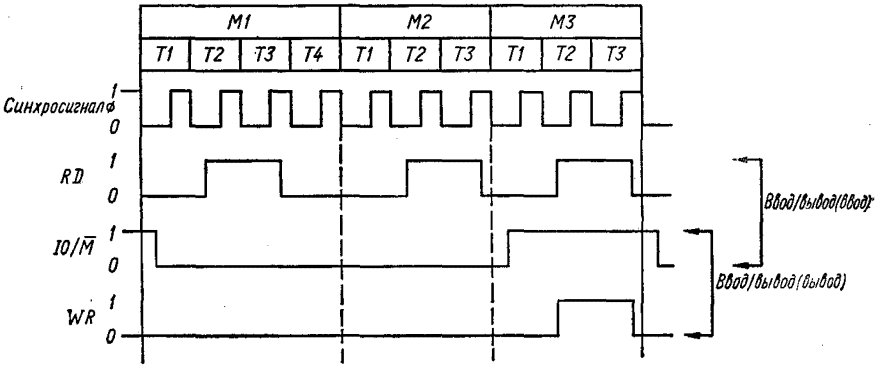


Рис. П2.10

менные диаграммы показаны на рис. П2.10. Для простоты на рисунке не показаны фронты и спады сигналов. Для более подробного ознакомления с диаграммами необходимо обратиться к литературе [1].

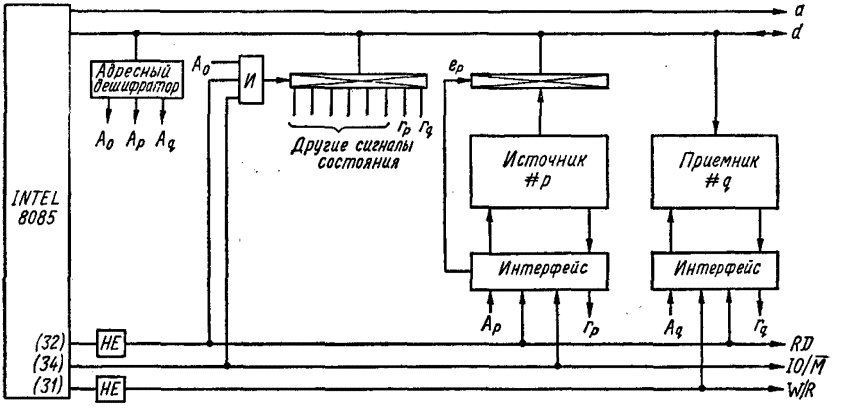


Рис. П2.11

Структурная схема системы проверки и пропуска на базе микропроцессора *In 8085* для одного источника и одного приемника данных показана на рис. П2.11. Сигнал $r_p = 1$, когда источник выдает данные. Аналогично $r_q = 1$, когда приемник готов принять данные. Сигнал e_p имеет три состояния и вырабатывается путем объединения по схеме *И* сигналов A_p, R_d и $I/\overline{OM}$.

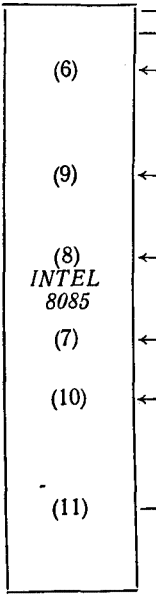
П2.4. СИСТЕМЫ ПРЕРЫВАНИЙ [1]

Микропроцессор *In 8085*, как уже указывалось, имеет 5 входов прерываний: *TRAP, RST7.5, RST6.5, RST5.5* и *INTR* (рис. П2.12). На эти сигналы не накладываются никакие временные ограничения — они могут появляться в лю-

бое время. Входы прерываний (контакты *6—11*, рис. *П2.12*) стробируются в течение последнего синхропериода выполняемой команды. Запросы прерываний имеют фиксированный приоритет, показанный ниже

TRAP	Высший приоритет
*RST*7.5	↓
*RST*6.5	
*RST*5.5	
INTR	Низший приоритет

TRAP — немаскируемое прерывание (повторный запуск, *RST*), используемое преимущественно в экстренных ситуациях. Этот сигнал должен быть высокого уровня в момент стробирования, однако, для того чтобы этот сигнал



Немаскируемое прерывание. Переход «0» → «1» вызывает передачу управления в микропроцессоре по адресу $000\,044_8$ после окончания текущей команды. Должен быть равен 1 в момент стробирования. Асинхронный

Переход «0» → «1» вызывает передачу управления по адресу $000\,075_8$ после окончания текущей команды. Асинхронный

Подача «1» вызывает передачу управления по адресу $000\,065_8$ после окончания текущей команды. Асинхронный

Подача «1» вызывает передачу управления по адресу $000\,055_8$ после окончания текущей команды. Асинхронный

Подача «1» прерывает выполнение программы в конце текущей команды. На шину данных в течение цикла обработки прерывания подается команда *CALL* или *RST*. Асинхронный

Этот сигнал генерируется в течение цикла следующего за приемом сигнала *INTR*. Он имеет те же самые временные характеристики, что и сигнал $\overline{RD}$ и используется вместо него

Рис. 2П.12

был воспринят в следующий раз, он сначала должен принять значение «0», а затем — опять «1».

Три прямых рестарта, *RST7.5, RST6.5* и *RST5.5,* могут быть каждый в отдельности программно замаскированы (т. е. запрещены) с помощью команды SIM (рис. П2.4). Отметим, что запрос *RST7.5* можно подать даже тогда, когда его маска установлена, а прерывания запрещены. На рис. П2.1 показано, что подача «0» на контакт *36* устанавливает все маски прерываний, т. е. запрещает прием прерываний через контакты *7, 8, 9* (рис. П2.12).

INTR используется как прерывание общего назначения, этот вход эквивалентен входу запроса прерывания микропроцессора *In 8080,* т. е. если *INTR* — единственный выставленный запрос прерывания и флажок разрешения прерывания *INTEFF* установлен, микропроцессор *In 8085* сбрасывает этот флажок и входит в цикл обработки прерывания, т. е. все действия выполняются так же, как в *In 8080.* Цикл обработки прерывания похож на цикл извлечения команды за двумя исключениями. Сигнал $\overline{INTA}$ устанавливается вместо $\overline{RD}$ (см. рис. П2.10). Адресные шины не несут информации. Когда $\overline{INTA}$ установ-

лен, логическая схема прерывания должна выдать для выполнения код операции. Хотя будет выполнена любая команда, но обычно подается команда CALL или RST. Это связано с тем, что обе эти команды вызывают засылку содержимого программного счетчика микропроцессора в стек перед переходом по новому адресу. Если это утверждение недостаточно ясно, то более подробное изложение читатель может найти в параграфе 5.1. Для *In 8080* использовалась команда RST (см. параграф 5.4), а для *In 8085* — команда CALL.

После получения кода операции в состоянии *M1·T3* (рис. П2.5) микропроцессор декодирует ее в следующем цикле *M1·T4* и определяет, что необходимы еще два байта. Далее *In 8085* выполняет два дополнительных машинных цик-

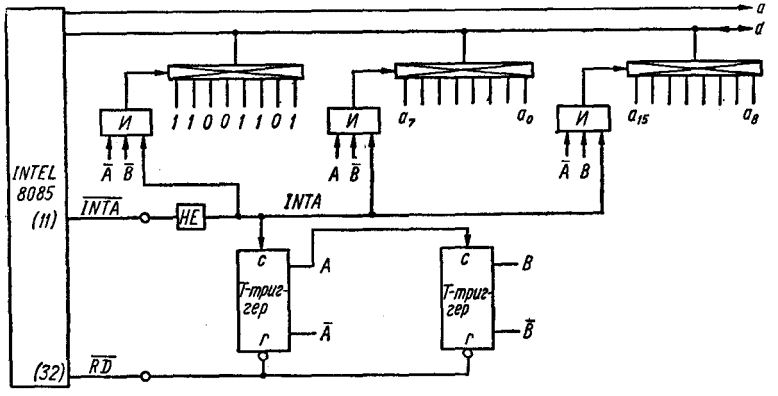


Рис. П2.13

ла, извлекая второй и третий байт. Как и в случае выборки кода операции, байты 2 и 3 выдаются на шину данных, когда сигнал *INTA* не равен 0. В течение цикла обработки прерывания содержимое программного счетчика не увеличивается.

В течение машинных циклов *M4* и *M5* микропроцессор засылает старший и младший байты содержимого программного счетчика в стек и помещает на их место два байта, принятые в циклах *M2* и *M3*. Благодаря этому происходит передача управления по адресу, указанному в команде CALL.

На рис. П2.13 показана простая схема для выдачи на шину данных команды CALL в течение цикла прерывания, состоящая из счетчика импульсов (до трех) и трех портов ввода/вывода. Вначале счетчик сбрасывается сигналом $\overline{RD}$. В течение цикла обработки прерывания после получения сигнала *INTR* (запрос прерывания) импульсы $\overline{RD}$ подавляются. Вместо них генерируются три импульса $\overline{INTA}$ с теми же временными характеристиками. Эти импульсы используются в качестве счетных в счетчике. Так как содержимое счетчика увеличивается по спаду синхроимпульса, то условия выдачи трех байтов команды CALL следующие:

$\overline{A}\overline{B}\ \overline{INTA}$ — выдача кода операции 11001101;
$\overline{A}B\ \overline{INTA}$ — выдача байта 2 (младшая половина адреса);
$A\overline{B}\ \overline{INTA}$ — выдача байта 3 (старшая половина адреса).

Счетчик сбрасывается следующим импульсом $\overline{RD}$.

П2.5. СПИСОК ЛИТЕРАТУРЫ

1. **MCS85 User's Manual** (Preliminary), Intel Corporation. 1976, 1977.

СОДЕРЖАНИЕ

Стр.

От редактора перевода 3
Предисловие автора 5
Введение 6
Глава 1. Логическое проектирование 11
1.1. Базовые определения 11
1.2. Оптимальное проектирование 12
1.3. Булева алгебра 12
1.4. Логические элементы 18
1.5. Гонки 20
1.6. Неиспользуемые состояния 21
1.7. Уменьшение числа состояний 21
1.8 Последовательностные уравнения 22
1.9. Асинхронные нетактируемые последовательностные схемы . . 26
1.10. Синхронные тактируемые последовательностные схемы . . . 29
1.11. Список литературы 33
Глава 2. Микропроцессоры 34
2.1. Описание микропроцессора 34
2.2. Состояние ожидания 38
2.3. Сигналы микропроцессора 45
2.4. Режимы работы микропроцессора 45
2.5. Полупроводниковая память 47
2.6 Порты ввода/вывода 50
2.7. Адресные дешифраторы 51
2.8. Интерфейсы 51
2.9. Методика проектирования 52
2.10. Процедура проектирования [1] 52
2.11. Список литературы 53
Глава 3. Системы ожидания/счета 54
3.1. Базовые определения 54
3.2. Концепция ожидания/счета 55
3.3 Системы ожидания/счета 55
3.4. Логическая схема ожидания/счета 60
3.5. Задачи и решения 67
3.6. Список литературы 92
Глава 4. Системы проверки и пропуска 92
4.1. Базовые определения 92
4.2. Системы проверки и пропуска 93
4.3. Растягивание синхроимпульса 94
4.4. Задачи и решения 95
Глава 5. Системы прерываний 105
5.1. Базовые определения 105
5.2. Системы прерываний 107
5.3. Флажки и их сортировка 109
5.4 Система прерываний микропроцессора *In 8080* 114
5.5. Чрезвычайные прерывания микропроцессора *In 8080* 118
5.6. Система прерываний микропроцессора *Mc 6800* 119
5.7. Экстренные прерывания микропроцессора *Mc 6800* 120

5.8. Задачи и решения 120
5.9. Список литературы 136
Глава 6. Системы прямого доступа к памяти (ПДП) 136
6.1. Базовые определения 136
6.2. Система ПДП 137
6.3. Интерфейс ПДП 138
6.4. Двухпроводной интерфейс 141
6.5. Логическая схема захвата цикла 144
6.6. Задачи и решения 146
Глава 7. Системы прямой передачи данных 149
7.1. Базовые определения 149
7.2. Системы ППД 150
7.3. Интерфейс ППД 151
7.4. Задачи и решения 155
7.5. Список литературы 159
Приложение 1. Устройства типа запрос/ответ 160
П1.1. Устройство запрос/ответ 160
П1.2. Логическая схема, чувствительная к фронтам 160
П1.3. Список литературы 162
Приложение 2. Микропроцессор *Intel 8085* 162
П2.1. Общие положения 162
П2.2. Системы ожидания/счета 165
П2.3. Системы проверки и пропуска 171
П2.4. Системы прерываний 172
П2.5. Список литературы 174

Д. ЗИССОС

Проектирование систем на микропроцессорах

Редактор Н. М. Корнильева
Оформление художника С. К. Данильченко
Художественные редакторы Л. А. Дикарев, В. С. Шапошников
Технический редактор С. В. Иванус
Корректор Н. Г. Петрик

Информ. бланк № 2134

Сдано в набор 05.02.81. Подписано в печать 13.11.81. Формат 60×90¹/₁₆. Бумага типогр. № 1. Гарн. лит. Печ. выс. Усл. печ. л. 11,0. Усл. кр.-отт. 11,0. Уч.-изд. л. 11,55. Тираж 10 000 экз. Зак. 1-459. Цена 1 р. 10 к.

Издательство «Техника», 252601, Киев, 1, ГСП, Крещатик, 5.

Отпечатано с матриц Киевской книжной фабрики на Харьковской книжной фабрике «Коммунист», 310012, Харьков-12, Энгельса, 11.

42

1р.10к.

Д. ЗИССОС

1р. 10к.